2015-06

# Optimization of source and receiver placement in multistatic sonar environments

Hof, Christoph

Monterey, California: Naval Postgraduate School


# NAVAL POSTGRADUATE SCHOOL

**MONTEREY, CALIFORNIA**

# THESIS

**OPTIMIZATION OF SOURCE AND RECEIVER PLACEMENT IN MULTISTATIC SONAR ENVIRONMENTS**

by

Christoph Hof

June 2015

| | |
|---|---|
| Thesis Advisor: | Emily M. Craparo |
| Second Reader: | Mümtaz Karataş |

THIS PAGE INTENTIONALLY LEFT BLANK

# REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188



| 1. AGENCY USE ONLY *(Leave Blank)* | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | 06-20-2015 | Master's Thesis 11-10-2014 to 06-20-2015 |

**4. TITLE AND SUBTITLE**

OPTIMIZATION OF SOURCE AND RECEIVER PLACEMENT IN MULTISTATIC SONAR ENVIRONMENTS

**5. FUNDING NUMBERS**

**6. AUTHOR(S)**

Christoph Hof

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Naval Postgraduate School
Monterey, CA 93943

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

N/A

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**

Approved for public release; distribution is unlimited

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(maximum 200 words)*

The usage of non-collocated sources and receivers in multistatic sonar networks (MSNs) has emerged as a promising area of opportunity in sonar systems. This thesis addresses point coverage sensing problems in MSNs, where a number of points of interest (targets) must be monitored. For detection we assume cookie-cutter sensors and a two-dimensional homogenous environment. Based on current algorithms and the theory of detection discs, we develop the new preprocessing algorithm LOC-GEN-II to determine possible locations for sources given a set of targets and receivers. The high efficiency of this algorithm is based on the greedy-like approach it is built upon and allows a significant reduction of computing time compared to a recent algorithm from the literature.

We also address the problem of optimally placing multiple sources and receivers for a given set of targets. Up to now, this problem was solved with the simplification of setting receivers randomly and placing only sources optimally. We develop LOC-GEN-II further into a two-step process of determining near optimal positions for receivers and sources successively. The procedure is implemented as a faster one-step solution and a slower iterative approach, which leads to better detection results. With this approach we show that the newly developed algorithms allow solution of multiple sensor placement problems in an acceptable time with significantly better detection results compared to the benchmark.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| Multistatic Sensor Network, Point Coverage Sensing, Optimization | 105 |
| | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UU |

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

**Approved for public release; distribution is unlimited**

# OPTIMIZATION OF SOURCE AND RECEIVER PLACEMENT IN MULTISTATIC SONAR ENVIRONMENTS

Christoph Hof
Lieutenant Colonel, German Army
Dr.-Ing., University of the German Armed Forces Hamburg, 2002

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN OPERATIONS RESEARCH**

from the

**NAVAL POSTGRADUATE SCHOOL**
**June 2015**

Author: Christoph Hof

Approved by: Emily M. Craparo
Thesis Advisor

Mümtaz Karataş
Second Reader

Robert Dell
Chair, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

The usage of non-collocated sources and receivers in multistatic sonar networks (MSNs) has emerged as a promising area of opportunity in sonar systems. This thesis addresses point coverage sensing problems in MSNs, where a number of points of interest (targets) must be monitored. For detection we assume cookie-cutter sensors and a two-dimensional homogenous environment. Based on current algorithms and the theory of detection discs, we develop the new preprocessing algorithm LOC-GEN-II to determine possible locations for sources given a set of targets and receivers. The high efficiency of this algorithm is based on the greedy-like approach it is built upon and allows a significant reduction of computing time compared to a recent algorithm from the literature.

We also address the problem of optimally placing multiple sources and receivers for a given set of targets. Up to now, this problem was solved with the simplification of setting receivers randomly and placing only sources optimally. We develop LOC-GEN-II further into a two-step process of determining near optimal positions for receivers and sources successively. The procedure is implemented as a faster one-step solution and a slower iterative approach, which leads to better detection results. With this approach we show that the newly developed algorithms allow solution of multiple sensor placement problems in an acceptable time with significantly better detection results compared to the benchmark.

THIS PAGE INTENTIONALLY LEFT BLANK

# Table of Contents


**1 Introduction** **1**

1.1 Motivation . . . . . 1

1.2 Literature Review . . . . . 3

1.3 Objectives . . . . . 10

1.4 Scope, Limitations, and Assumptions . . . . . 11

1.5 Contributions and Outline . . . . . 12

**2 Optimization of Candidate Point Determination Using Detection Discs and Non-Dominated Point Sets** **13**

2.1 Problem Formulation . . . . . 13

2.2 Detection Discs for Optimal Source Placement . . . . . 13

2.3 Implementation of Preprocessing . . . . . 16

2.4 Enhanced Preprocessing Algorithm . . . . . 19

2.5 Computational Comparison of LOC-GEN and LOC-GEN-II . . . . . 25

2.6 Connection of Preprocessing and Optimal Source Location . . . . . 32

2.7 Investigation of Number of Elements in the Candidate Point Lists . . . . . 35

**3 Improved Placement of Sources and Receivers** **45**

3.1 Source and Receiver Positioning Utilizing Detection Discs . . . . . 45

3.2 Near Locally Optimal Placement of Receivers and Sources . . . . . 52

3.3 Exact Locally Optimal Placement of Receivers and Sources . . . . . 59

**4 Conclusion** **65**

**Appendix A Calculation of Intersection Points of Detection Discs** **69**

**Appendix B MATLAB Implementation Of LOC-GEN-II** **71**

**Supplementals** **81**

**References** **83**

**Initial Distribution List** **85**

# List of Figures


Figure 1.1 Geometry of Sonar Detection . . . . . 2

Figure 1.2 Multistatic Sonar Detection . . . . . 4

Figure 1.3 Cassini Ovals . . . . . 6

Figure 1.4 Detection Discs . . . . . 7

Figure 1.5 Receiver Discs . . . . . 10

Figure 1.6 Source Discs . . . . . 10

Figure 2.1 Optimal Point Positions . . . . . 15

Figure 2.2 Possible Optimal Location Determination . . . . . 15

Figure 2.3 Optimal Source Placement Algorithm . . . . . 16

Figure 2.4 Nearly Optimal Source Placement Algorithm . . . . . 17

Figure 2.5 Source Location Reduction . . . . . 18

Figure 2.6 Enhanced Possible Optimal Location Determination . . . . . 19

Figure 2.7 Optimal Point Positions . . . . . 21

Figure 2.8 Detection Discs with no Intersections . . . . . 22

Figure 2.9 Determination of Optimal Source Positions . . . . . 25

Figure 2.10 Final Candidate Point Location for 15 Targets and 15 Receivers . 26

Figure 2.11 Final Candidate Point Location for 150 Targets and 50 Receivers 27

Figure 2.12 Iterative Reduction of Elements in Candidate Point List . . . . . 28

Figure 2.13 Candidate Point Reduction—Comparison of Algorithms—25 Receivers . . . . . 29

Figure 2.14 Candidate Point Reduction—Comparison of Algorithms—100 Receivers . . . . . 29

Figure 2.15 Candidate Point Reduction—Computing Time—LOC-GEN-II . . 31
Figure 2.16 Candidate Point Reduction—Comparison of Algorithms—Large Number of Targets . . . . . 32
Figure 2.17 Reduction of Preprocessing and Optimization Time—Overview . 34
Figure 2.18 Reduction of Overall Computing Time—Overview . . . . . 35
Figure 2.19 Final Candidate Location List . . . . . 36
Figure 2.20 Final Candidate Points Distribution—Overview . . . . . 37
Figure 2.21 Final Candidate Points Distribution—Parts . . . . . 37
Figure 2.22 Regression Fit for Number of Candidate Points . . . . . 40
Figure 2.23 Final Candidate Points—Simulation and Fit . . . . . 41
Figure 2.24 Final Candidate Points—Varying Size of Range of the day . . . . 43
Figure 2.25 Final Candidate Points—Varying Size of Range of the day . . . . 44

Figure 3.1 Adapt-LOC to Find Best Places for Sources and Receivers . . . . 46
Figure 3.2 Improved Positioning of Receivers and Sources for 150 Targets using Adapt-LOC . . . . . 49
Figure 3.3 Improved Positioning of Receivers and Sources for 200 Targets Using Adapt-LOC . . . . . 51
Figure 3.4 Sensor Placement for Partially Random and Improved Position Determination . . . . . 52
Figure 3.5 Sensor Placement for Random and Improved Position Determination . . . . . 53
Figure 3.6 Iterative Positioning of Receivers and Sources for 100 Targets . . 54
Figure 3.7 Iterative Positioning of Receivers and Sources for 200 Targets . . 57
Figure 3.8 Steps for Iterative Positioning of Receivers and Sources . . . . . 57
Figure 3.9 Computing Time for Different Positioning Types of Receivers and Sources . . . . . 58

Figure 3.10 Locally Optimal Positioning of Receivers and Sources for 100 Targets . . . . . 60

Figure 3.11 Locally Optimal Positioning of Receivers and Sources for 200 Targets . . . . . 61

Figure 3.12 Locally Optimal Iterative Positioning of Receivers and Sources . 62

Figure 3.13 Computing Time for Iterative Exact Locally Optimal Positioning of Receivers and Sources . . . . . 63

THIS PAGE INTENTIONALLY LEFT BLANK

# List of Tables


| | | |
|---|---|---|
| Table 2.1 | Example of Dominated Location Elimination . . . . . | 18 |
| Table 2.2 | Increase in Number of Candidate Points with Rising Target Number | 39 |
| Table 2.3 | Computing Time and Remaining Candidate Locations for High Target Numbers . . . . . | 42 |
| Table 3.1 | Detection of Targets Using Improved Source and Receiver Placement | 47 |
| Table 3.2 | Relation Between Number of Different Sensors, Radius of Target Detection Discs, and Detection Rate . . . . . | 50 |
| Table 3.3 | Target Detection Probability for Different Sensor Placement Methods | 56 |

THIS PAGE INTENTIONALLY LEFT BLANK

# List of Acronyms and Abbreviations

| | |
|---|---|
| **ASW** | anti-submarine warfare |
| **AoI** | area of interest |
| **CP** | center point |
| **DiBS** | divide best sector |
| **DD** | detection disc |
| **DDS** | detection disc set |
| **GAMS** | general algebraic modeling system |
| **HVU** | high value unit |
| **ILP** | integer linear program |
| **IP** | intersection point |
| **MSN** | multistatic sonar network |
| **MODD** | mutually overlapping detection disc |
| **MODDS** | mutually overlapping detection disc set |
| **NATO** | North Atlantic Treaty Organization |
| **NPS** | Naval Postgraduate School |
| **NOLH** | nearly orthogonal Latin hypercube |
| **RD** | receiver disc |
| **RDC** | range of the day circle |
| **RoD** | range of the day |
| **SD** | source disc |
| **TL** | transmission loss |

THIS PAGE INTENTIONALLY LEFT BLANK

# Executive Summary

For decades, information from sonar systems has been used to detect underwater threats in anti-submarine warfare (ASW). Due to increased tensions between Russia and the North Atlantic Treaty Organization (NATO), detection of underwater threats has become important in recent times again, especially the usage of active multistatic sonar networks (MSNs) with separated sources and receivers. These systems have various advantages compared to conventional monostatic systems like flexibility, adaptability, more complicated countermeasures for the opponent, and lower costs. However, the mathematical relations for multistatic systems are more complex, which makes the solution of the layout of optimal detection problems much harder.

This thesis addresses point coverage problems in which a number of discrete locations are to be monitored. We assume a cookie cutter sensor in a two-dimensional homogenous environment. We neglect target aspect dependency and blind zones of sonar systems, and we do not perform target tracking.

First we review the concept of detection discs (DDs) for optimal sensor positions. Based on the detection range of a sensor—range of the day (RoD)—these are circles around targets with radii related to the RoD and the nearest sensor. Intersections or center points of these DDs are possible optimal positions for sensors. Additionally, optimal sensor positions must always be at non-dominated points. These points allow detection of the same or more targets than all other points, but never fewer or different ones.

The benchmark preprocessing algorithm to determine optimal sensor locations, which is based on this concept and was developed by Craparo and Karataş (2014), compares all possible sensor locations with each other. Due to the high complexity of this algorithm, computing times become unacceptably large with extensive problems. We develop a new preprocessing algorithm (LOC-GEN-II) based on an iterative greedy-like approach. Compared to the benchmark algorithm, it diminishes complexity and computing time significantly by reducing the possible candidate points and thus the remaining computing operations in each iteration step.

Following preprocessing, the optimal solution for sensor positions must be done. Exact op-

timal algorithms (OPT-LOC) and near optimal algorithms (Greedy-LOC), both developed by Craparo and Karataş (2014), provide a benchmark. We investigate the possibility to reduce overall computing time by varying the handover from preprocessing to optimal sensor positioning. This is realized by terminating the preprocessing at earlier stages in order to not investigate all possible candidate positions. LOC-GEN-II allows this processing based on the greedy-like approach it uses for candidate point selection. Reduction of computing times is possible, but only in subordinate magnitude.

In the following subsection, we investigate the number of final candidate points in the reduced candidate point list. We develop a relationship between number of targets and number of final candidate points using regression. The class of the candidate point (intersection point or center point of DD) is the pivotal criterion for the trend of the preprocessing time. Increasing target numbers lead to increasing numbers of final candidate points. Analyzing the influence of the detection range (RoD) leads to the interesting result of decreasing the number of candidate points for large values of RoD.

In the second part of this thesis, we develop two algorithms to deploy both kinds of sensors for a given set of targets. The benchmark is an algorithm developed by Craparo and Karataş (2014) which allows optimal positioning of sources for a given set of fixed targets and receivers. Based on the general idea of LOC-GEN-II, we use the concept of DD to solve the problem in a two-step or iterative approach. For a set of targets, our first algorithm, Adapt-LOC, calculates DDs to determine locations for the receivers, which are then used to compute optimal source locations. Compared to the benchmark algorithm, Adapt-LOC delivers considerably better detection results. We then develop Iter-LOC, an iterative algorithm to place both types of sensors locally optimally. This leads to another improvement of detection probability. For both Adapt-LOC and Iter-LOC, we compare near optimal greedy solutions with locally optimal integer linear programs (ILPs) solutions, and we find that ILP clearly outperforms the greedy approach.

Craparo, E. M., & Karataş, M. (2014). *Sensor optimization in multistatic underwater sensor networks*. (submitted manuscript, in review)

# CHAPTER 1:
# Introduction

## 1.1 Motivation

Sonar systems have been in use in underwater warfare to detect submarines for decades. Throughout this time, development of the systems proceeded and also found applications in non-military fields (Urick, 1983, p. 2). Researchers distinguish between active and passive sonar systems as well as between monostatic and multistatic systems. Active sonar systems are the main systems used in anti-submarine warfare (ASW). Due to increased tensions between NATO and Russia in recent times, the threat of underwater attacks (as physical attack or as espionage) increased significantly, thus making detection of underwater targets very important today (Daerden, 2015; Braw, 2014; Marcus, 2014).

An active sonar system consists of a source and a receiver. The source sends out a pulse of underwater sound (ping), which is reflected by the underwater environment and possibly a target in the area. The reflected signal is detected by a receiver which makes it possible to determine information about the objects in the vicinity, including their locations. In a monostatic sonar system, source and receiver are collocated and form a so-called post (Ozols & Fewell, 2011, p. 5). The principle can be seen in Figure 1.1 (left). In a multistatic sonar network (MSN), source and receiver are not necessarily collocated (see Figure 1.1, right), which has numerous advantages compared to the monostatic case, although such systems are mathematically far more complicated. Advantages of MSNs are:

- **Reduced costs:** Washburn and Karataş (2015, p. 1) mention that receivers cost less than sources, which can reduce the overall cost of the system, because one source can be used in combination with several receivers. Also, in its fiscal year 2015 budget estimates, United States Navy (2014, p. 23) underlines this with numbers, stating a multistatic source to be five times more expensive than a sonar receiver. Because of reduced military budgets and further reductions planned, cost optimization becomes more and more important nowadays (Simeone, 2014).
- **Countermeasures:** Receivers do not send out underwater signals; thus, they do not

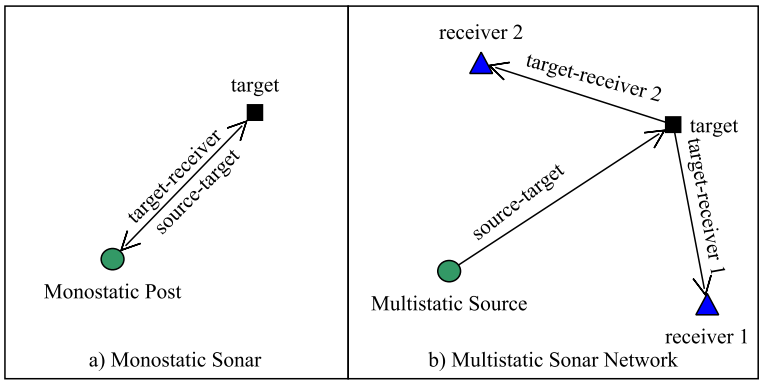


Figure 1.1: **Geometry of Sonar Detection**—left: monostatic case; right: multistatic case

reveal their positions to the possible target. This makes countermeasures of the target against the receivers more difficult, because the receivers are very hard to detect. In contrast, a source (and so also a sonar post as used in the monostatic case) will always be detected by the target by sending out its first ping.

- **Flexibility:** In a monostatic case researchers are fixed to the physical combination of source and receiver, whereas in a MSN, the receivers can, for example, be a field of sonobuoys fielded in an area of interest (AoI), and the source could be an active source, dipped via helicopter at specific points in the AoI or a ship, which sends out pings. This makes MSNs much more flexible and adaptable to solving specific detection problems.
- **Precision:** Coon (1997) shows that fusion of target information gathered by different receivers and resulting from only one ping of one source leads to an increased precision of detection. Additionally, this data fusion can lead to reduced "false alarms that monostatic active systems are normally prey to" (Washburn & Karataş, 2015, p. 1).

The complications with MSN arise due to the different geometry in comparison to the monostatic case, as can be seen in Figure 1.1. In the monostatic case the detection probability depends mainly on the distance between post and target. In case of a multistatic constellation, the distances between target and source as well as target and receiver are relevant (Fewell & Ozols, 2011, p. 8). This results in the fact that determination of optimal places for sources and receivers for a given fixed set of targets becomes mathematically complicated.

## 1.2 Literature Review

Multistatic sensor networks are a wide area of research for radar detection of targets in the air and near the surface or sonar detection of targets under water. We will only regard the sonar detection part in this thesis, but the relation to radar detection problems has to be mentioned. Multistatic radar detection for point and barrier coverage can be found in Johnsen and Olsen (2006) and Gong, Zhang, Cochran, and Xing (2013).

Two main questions occur related to sonar research: Which problem do we want to solve, and are we interested in a single point of time or a series of looks into the AoI. For the first question three main detection problems can be found in literature: area coverage, barrier coverage and point coverage problems. Many research projects deal with area coverage problems. Ngatchou, Fox, and El-Sharkawi (2006) optimize the coverage of an area while minimizing the number of required sensors. Ozols and Fewell (2011) give a detailed study of different geometric patterns of sources, receivers, and posts to find the optimal coverage of the whole AoI. DelBalzo, McNeal, and Kierstead (2005) and DelBalzo and Stangl (2009) focus on the optimal placement of sonobuoys in an area and the optimal path to bring them to their locations. For the second question a large area of research deals with the tracking of targets and the automated following of a tracked target (and so the investigation of time series of detections). This is challenging for the multistatic case, as can be seen in Coon (1997), Coraluppi and Grimmet (2003), Fewell and Ozols (2011), Orlando and Ehlers (2011), and Ozols and Fewell (2011).

For this thesis we focus on point coverage problems and a single glance at one point of time. Our focus lies in the optimal placement of sources and receivers to cover a given set of targets in a distinct area.

### 1.2.1 Multistatic Detection Theory

The detection of a multistatic sonar system is based on the transmission loss (TL) of the ping on its way from the source to the target (traversing distance $d_{t,s}$) and then from the target to the receiver (traversing distance $d_{t,r}$). It can be calculated as follows:

$$TL = 20 \cdot log(d_{t,s} \cdot d_{t,r}) + \alpha \cdot (d_{t,s} + d_{t,r}) \approx 20 \cdot log(d_{t,s} \cdot d_{t,r}) = 20 \cdot log(\rho^2_{t,s,r}) \qquad (1.1)$$

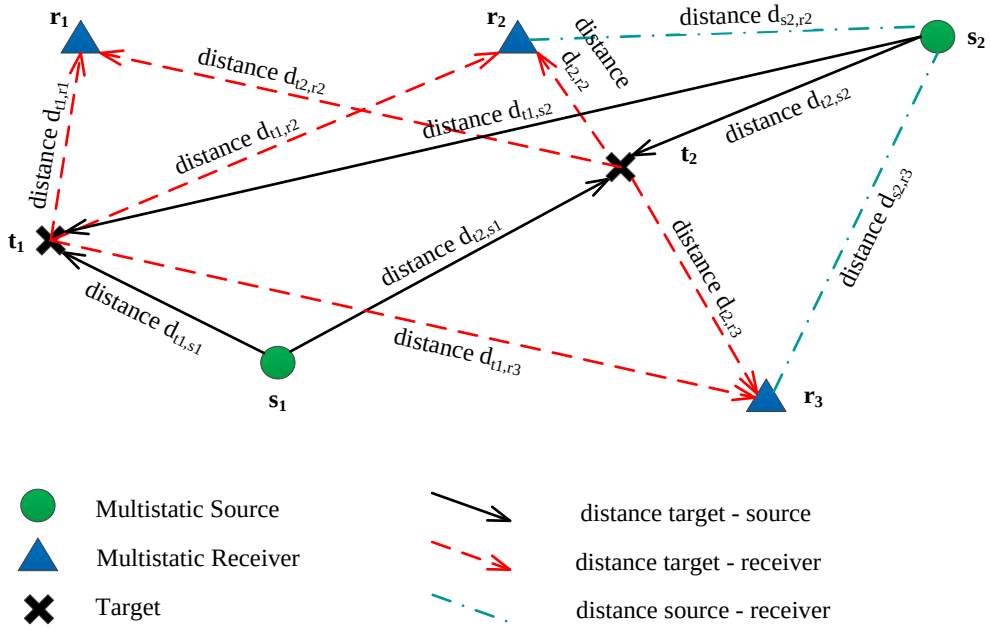


Figure 1.2: **Geometry of Multistatic Sonar Detection**—The signal is sent by the sources ($s_1$ and $s_2$) and reflected by the targets ($t_1$ and $t_2$). The distances between sources and receivers are shown for two cases, but they do not primarily affect the detection.

where $\rho_{t,s,r}$ is referred to as the *equivalent monostatic range*, because it can be used to determine the probability of detection of the multistatic configuration using "a monostatic detection probability curve" (Ozols & Fewell, 2011, p. 2). For more information see Craparo & Karataş, 2014; Kuhn, 2014; Ozols & Fewell, 2011; and Fewell & Ozols, 2011.

The geometry of a small MSN can be seen in Figure 1.2, which illustrates why multistatic sonar geometry becomes mathematically challenging. In this case only two sources, two targets, and three receivers are shown, but even with this small configuration the geometric possibilities for detection become numerous.

Different monostatic detection curves are used in literature, such as Fermi-shape, exponential-shape or cookie-cutter. An overview of these can be found in Kuhn, 2014, p. 4. For this thesis the assumed detection curve of the sensor has cookie-cutter shape, which means perfect detection within a specific range ($\rho_0$) and no detection outside. This leads to the

detection probability of a target as in Equation 1.2:

$$p_{t,s,r} = \begin{cases} 1 & \text{if } \rho_b \leq \sqrt{d_{t,s} \cdot d_{t,r}} = \sqrt{\rho_{t,s,r}^2} \leq \rho_0, \\ 0 & \text{otherwise,} \end{cases} \tag{1.2}$$

Here, $\rho_b$ is the so-called *blind zone* of the sensor where no detection is possible due to the fact that sent and reflected signals reach the receiver almost at the same time, and $\rho_0$ is the range of the day (RoD); every target up to this distance from the monostatic sonar post would be detected by the sensor. Bowen and Mitnick (1999) and Fewell and Ozols (2011) make detailed calculations including this blind zone, which makes the solution of multistatic sonar problem much more complex. To focus on the main topic and reduce the complexity of the modeled system, blind zones will not be considered in this thesis. Equation 1.2 implies that target $t$ is detected by receiver $r$ due to a ping of source $s$ with probability one if $\rho_{t,s,r} \leq \rho_0$. Keeping in mind that we can have many sources and receivers, the combination of possible detection triplets for a specific target $\tilde{t}$ becomes large, and the overall detection probability is the following (Kuhn, 2014, p. 4):

$$P_{\tilde{t}} = \max_{(s,r) \in S \times R} P_{\tilde{t},s,r} \tag{1.3}$$

where in our study, one receiver-source pair detecting $\tilde{t}$ is sufficient for overall detection.

For the multistatic case, the contour shapes around the source and receiver for a fixed $\rho_0$ form so-called *Cassini ovals* (see Figure 1.3; Fewell & Ozols, 2011, p. 5). Due to the assumption of a cookie-cutter sensor, all targets inside the contour can be observed. The shape changes from a circle to an ellipse to a dog-bone shape with increasing distance between source and receivers up to $d_{s,r} \leq 2 \cdot \rho_0$. For $d_{s,r} \geq 2 \cdot \rho_0$ two separated, oval-shaped regions of detection remain. This case, in particular, shows advantages of MSNs compared to monostatic situations, because observation of spatially disjoint areas with one system pair is possible. A system with several receivers located around a single source in different distances has an opportunity to cover a large AoI (possibly disjoint AoIs) with a minimum numbers of elements. But it also makes clear that the geometry of detection in MSNs is much more complex compared to a monostatic network.

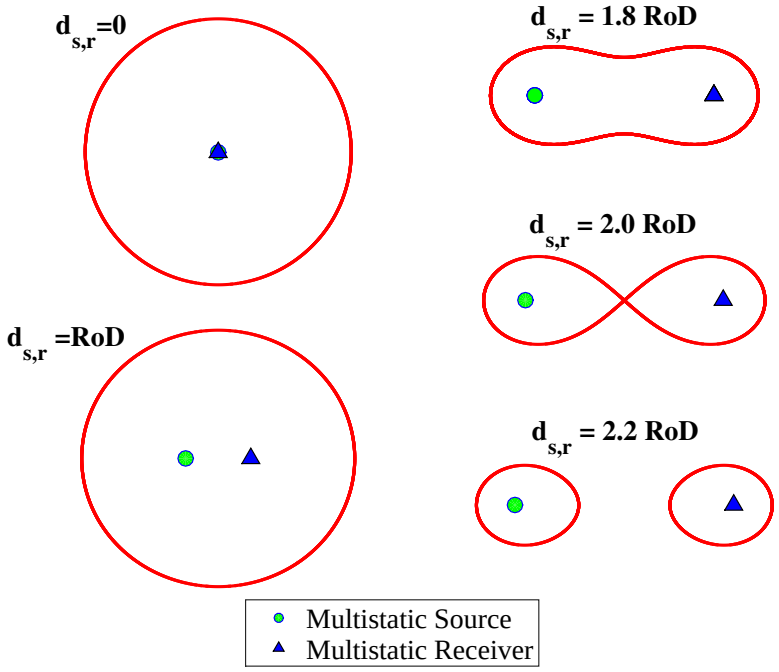


Figure 1.3: **Cassini Ovals for Different Values of $d_{s,r}$ with RoD**$=\rho_0$—$d_{s,r}=0$ represents the monostatic case and the configuration for a sonar post. Inside the contour line the inequality $d_{t,s}\cdot d_{t,r}\leq\rho_0^2$ is fulfilled; a target placed inside the shape or on the contour line can be detected by the source-receiver pair.

Some targets in an AoI can contain high value units (HVUs) and may therefore have a higher priority of observation. To take this into account while determining the overall probability of detection of the set of targets $T$, one can introduce values related to the targets ($w_t$). The resulting overall measure of performance and the objective value of the problem (Craparo & Karataş, 2014, p. 9) becomes, in relation with 1.3,

$$P_{max}=\sum_{t\in T} w_t\cdot P_t. \tag{1.4}$$

### 1.2.2 Concept of Detection Discs for Multistatic Sonar Systems

As shown in section 1.2.1 a cookie-cutter sensor has perfect detection when $\rho_{s,t,r}\leq\rho_0$. This opens the opportunity to use "detection discs" (see Craparo & Karataş, 2014, p. 6)

with radius $\rho_t$ which represents the area around a target $t$, where a placed source-receiver pair would definitely detect the target. In the case in which only positions of targets are given, Kuhn (2014, p. 14) introduces these as range of the day circles (RDCs) with radii $\rho_0$. They represent the possible locations of the collocated sensors (posts) for targets to be detected. An example can be seen in Figure 1.4.

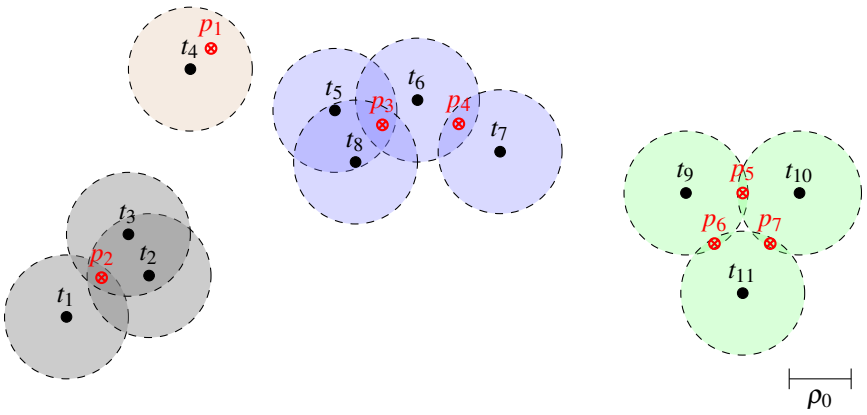


Figure 1.4: Detection Disc Example—Detection discs for a set of several targets, where the sensor has a cookie-cutter shape with detection radius $\rho_0$. This configuration leads to the following detection discs, where sensors should be placed to detect the targets $t_i$ at the locations $p_j$.

We first explain the theory for collocated posts. Every post (e.g., $p_1$) positioned inside a detection disc (DD) will detect at least one target, for example, a sensor inside the red circle will detect definitely target $t_4$. But it is also possible that one post can detect multiple targets, as in case of the triple $\{t_1,t_2,t_3\}$. The RDCs of three targets have a small region in common, so that by positioning $p_2$ inside this region, we will detect all three of them. The blue group of targets can be separated into two parts, $\{t_5,t_6,t_8\}$ and $\{t_6,t_7\}$. Each of the subsets has its own common region, but the two subsets have no overall common region.

To partition the RDCs in such configurations, one can use the notation of clusters $G \subseteq T$. "A cluster $G \subseteq T$ is a maximal set of targets, where the RDCs of all targets $t \in G$ have at least one point in common" (Kuhn, 2014, p. 16). With this example Figure 1.4 can be divided in the following clusters or subsets:

$$G_1 = \{t_4\}, \quad G_2 = \{t_1,t_2,t_3\}, \quad G_3 = \{t_5,t_6,t_8\}, \quad G_4 = \{t_6,t_7\},$$
$$G_5 = \{t_9,t_{10}\}, \quad G_6 = \{t_9,t_{11}\}, \quad G_7 = \{t_{10},t_{11}\}.$$

For each cluster one sensor is needed to detect all targets in that cluster. For the whole set of targets $T$, the number of clusters and therefore the number of mutually exclusive subsets $G_i$ is an upper bound on the number of sensors needed to detect all targets. Coming back to the example, the blue group with the two clusters $G_3$ and $G_4$ needs two sensors ($p_3$ and $p_4$) so all targets $\{t_5, t_6, t_7, t_8\}$ can be detected. Despite the whole group being connected, the related clusters have only target $t_6$ in common. Thus $G_3$ and $G_4$ are subsets not dominating each other (this becomes important in Chapter 2). The green-colored group consists of three mutually independent clusters $\{G_5, G_6, G_7\}$. In order to detect the three targets of this subset, only two of the three possible sensor positions ($p_5$, $p_6$, $p_7$) are sufficient. If one picks, for example, $p_5$ and $p_6$, the first sensor allows detection of $\{t_9, t_{10}\}$, the second of $\{t_9, t_{11}\}$, where $G_5 \cap G_6$ contains all three targets of the green group. This example shows a case where all clusters of the groups consist of different elements, but not all of the clusters must be considered for a sensor position, because the combination of some of the clusters ensures the covering of all elements.

In a real world situation one may have more information about the setup of AoI; for example, one may know the position of targets $t_i \in T \; \forall i$ and receivers $r_j \in R \; \forall j$. This is the setup which Craparo and Karataş (2014) use to determine optimal source positions, and, moreover, it is the starting configuration for the work of this thesis. In such a situation one can define DDs for every receiver-target combination. These receiver discs (RDs) have their centers at the target positions and radii of

$$\Delta_{t,r} = \frac{\rho_0^2}{d_{t,r}}. \tag{1.5}$$

Equivalently to RDCs for posts, every source which is placed inside or on the edge of this RD leads to a detection of the target by at least one receiver. Since we assume perfect detection characteristic of the sensors and only one receiver of the whole set $R$ must be able to detect a specific target $t$, the largest value of $\Delta_{t,r}$ for any receiver determines the maximal possible detection range; see Equation 1.6. This is achieved when the distance between target and receiver is minimal ($d_{t,r^*(t)} = \min_{\forall i} d_{t,r_i}$), leading to the following (Craparo &

Karataş, 2014, p. 6);

$$p_t = \begin{cases} 1 & \text{if } d_{t,s^*(t)} \leq \rho_0^2 / d_{t,r^*(t)} \\ 0 & \text{otherwise} \end{cases} \qquad \forall t \in T. \tag{1.6}$$

Equation 1.6 reflects the detection probability of a target $t$, given its nearest source $s^*(t)$ and receiver $r^*(t)$. Similarly, one can start from a setup with given targets and sources, what leads to source discs (SDs), which represent the places where receivers have to be placed to detect a given set of of targets and sources. In this case the detection probability is calculated via

$$p_t = \begin{cases} 1 & \text{if } d_{t,r^*(t)} \leq \rho_0^2 / d_{t,s^*(t)} \\ 0 & \text{otherwise} \end{cases} \qquad \forall t \in T. \tag{1.7}$$

These relations are visualized in Figure 1.5 and Figure 1.6. The first figure shows a setup with four targets and two receivers. Around every target, its RDCs can be seen and therefore the set of all locations for posts to detect the targets. Intersecting RDCs show areas where one post can detect more than one target. For example, this is the case between $t_1$ and $t_2$, where $r_1$ is located. Moreover, one can see the RDs related to the two receivers in blue and black. In this example the receiver discs of $r_1$ associated with $t_1$ and $t_2$ overlap, which means one source $s_k$ placed somewhere in the intersection region allows detection of both targets. But the corresponding RDs belonging to $t_3$ and $t_4$ neither overlap with each other nor with the discs of $t_1$ and $t_2$. Thus a detection of all targets using only receiver $r_1$ is not possible. When we look now at receiver $r_2$, we observe a common region of the RDs corresponding to $t_2$, $t_3$, and $t_4$. Placing a source in this (small) intersection region would guarantee a detection of these three targets using only receiver $r_2$. But there is no intersection with the disc of target $t_1$, therefore a complete detection of all targets using $r_2$ is not possible either. Furthermore, there is a region where all receiver discs (related to both receivers, $r_1$ and $r_2$) overlap. If one places a source in this region, for example, at the location of the green circle, one would be able to detect all four targets using only one source and the two given receivers $r_1$ and $r_2$.

Figure 1.6 shows the corresponding SDs for the target configuration of Figure 1.5 and the

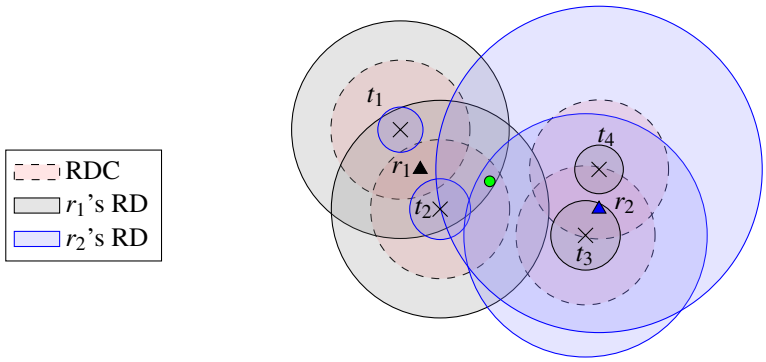


Figure 1.5: **Receiver Discs for a Given Set of Targets $t_i$ and Receivers $r_j$**—RDCs are depicted as red discs, RDs of $r_1$ as black discs, RDs of $r_2$ as blue discs. Intersection regions of RDs represent areas which allow detection of correspondent targets when placing a source in these regions.

placement of the source $s$ at the location of the green circle. As can clearly be seen, the associated SDs show two overlapping regions for each pair of targets. Positioning two receivers in these regions, let us say at the positions of the triangles, allows the detection of all four targets with a set of one source and two receivers.

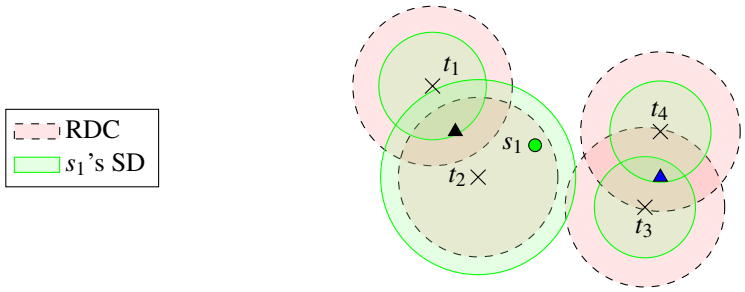


Figure 1.6: **Source Discs for a Given Set of Targets $t_i$ and Sources $s_j$**—RDCs are depicted as red discs, SDs of $s_1$ as green discs. Intersection regions of SDs represent areas which allow detection of correspondent targets when placing a source in these regions.

## 1.3 Objectives

Detection and observation of a given set of targets is the focus of this thesis. This can be, for example, oil platforms, offshore wind generators, or other stationary maritime facilities of high value. To protect these HVUs a point coverage problem with stationary targets/objects has to be solved. The aim is to observe all these targets with the minimum set of multistatic sensors (sources and receivers). Therefore we must find optimal positions for these sensors

under the restriction that we have only a specific set of sensors available for a given observation problem. Starting from the basic principles of multistatic sonar detections, two main questions are covered in this thesis:

- Using DDs, how can the search be optimized for the optimal positions of a set of sources for a target and receiver configuration? How can clusters and subsets of target be applied for a better preprocessing of data? Can different algorithms in the preprocessing be combined to make it more efficient? Are there possibilities to estimate the number of non-dominated sets for a given configuration of sensors?
- Starting from the problem solved in first bullet point, how can we expand the problem statement to the situation that only target positions are known? How can one iteratively solve the problem of optimal source and receiver placement? Is placement at one time or successive placement more advantageous, and is one of the two ways impossible to solve in an admissible computing time?

## 1.4 Scope, Limitations, and Assumptions

For simplicity we assume a two-dimensional setup of the problem with homogeneous environmental conditions. The positions of the targets are assumed to be known and fixed. The targets may have different values. A weight factor $w(t)$ is used to determine the importance of a specific target, with $w(t) \in [0,1]$; higher values mean higher priority. Additionally, for the first part of the analysis the receivers positions are also given and fixed. Once a source is placed optimally according to the algorithm, it is also set as stationary.

The sensors have cookie-cutter probability function of detection with RoD$= \rho_0$. For detection it is sufficient that one source-receiver pair can detect the target; although multiple detections are possible, they will not be considered in this thesis. Also, for simplicity, the blind zones of the sensors are neglected in this study. The detection results reflect a snapshot of the situation assuming that all sources send out one ping at a specific point of time and the resulting reflections are used by the receivers to determine whether an object is present at each target. Target tracking is not taken into account in this study.

## 1.5 Contributions and Outline

In Chapter 2 we show the theory of dominated and non-dominated sets of possible source positions for a given set of targets and receivers using DDs. The theory, developed by Craparo and Karataş (2014), is used to generate a new algorithm for preprocessing, which is based on a greedy-like approach to eliminate dominated sets. This can be found in Sections 2.2-2.5. In Section 2.6 the handover between preprocessing and optimization part of the calculation is investigated to optimize computing time further. Additionally, as the last part of Chapter 2, the relationship between number of targets, sensors and possible sensor locations is investigated in Section 2.7.

The second part of the study in Chapter 3 uses these results to build a solution approach for the more general problem of having only the targets fixed in the AoI. This is done in Section 3.1. We first describe a two-step process that involves successively placing sources and then, based on their position, receivers (or vice versa). This process can be found in Section 3.2. Another solution uses the iterative repetition of this process until a locally optimal solution is reached; this can be found in Section 3.3. Finally, we compare these two methods to evaluate their performance for different problem configurations.

# CHAPTER 2:
# Optimization of Candidate Point Determination Using Detection Discs and Non-Dominated Point Sets

## 2.1 Problem Formulation

The determination of the optimal position of a set of sources for a given set of targets and receivers can be done using different approaches. Kuhn (2014, p. 23) used for the bistatic case the iterative divide best sector (DiBS) algorithm. This "partitions [in each iteration step] the area of possible solutions into sectors" and selects the one with the highest upper bound of an objective function. This dividing and selecting is repeated, until a termination condition (sector size or optimality gap) is fulfilled. Problems arise with multiple sensors (sources), since the complexity of the algorithm to find optimal positions becomes computationally not solvable in acceptable time (Kuhn, 2014, p. 39). Washburn and Karataş (2015) compute the detection probability using random positions for both sources and receivers. So they do not optimally place the sources, but develop analytical functions to approximate the detection probability for a given set of sources and receivers.

For this section of the thesis we assume a quadratic AoI with randomly placed targets $t \in T$ and receivers $r \in R$. The targets have weights $0 \leq w_t \leq 1$ related to their importance, where 0 means totally unimportant and 1 highly important. The task is to find the optimal positions of sources $s \in S$, which allow the detection of the most targets $t$. The numbers of elements in $T$ and $R$ and $S$ can vary. Craparo and Karataş (2014, p. 6) are the first to develop an algorithm for the optimal solution for this MSN problem.

## 2.2 Detection Discs for Optimal Source Placement

The theoretical background, proofs, and general optimization algorithms written in this subsection are all based on the work of Craparo and Karataş (2014); the remainder of this section is also based on their work.

The probability of detecting target $t$ using its nearest source $s^*(t)$ and receiver $r^*(t)$ can be found in 1.6. This equation implies the existence of disc-shaped regions $\delta_t$ around every

target position. When at least one source is placed inside a specific disc $\delta_{\tilde{t}}$, the dependent target $\tilde{t}$ will be detected by at least one source-detector pair. The DDs can be combined in a detection disc set (DDS) with $D = \{\delta_1, \delta_2, ...\delta_T\}$ and the placement of sources in overlapping regions of DDs lead to detection of multiple targets with a single source.

Using this, optimal source position can be found by

1. Identification of maximal subsets of targets detectable by each source
2. Identification of possible candidate position for each subset
3. Maximization of detected number of targets by selection of source locations from possible candidate positions

The main computational challenges can be found in steps one and two, because they are of high complexity. To solve these problems, we need the following elements:

- $\overline{D} \subseteq D$: mutually overlapping detection disc set (MODDS): a set of detection discs $\overline{D}$ is mutually overlapping if $\exists x \in$ AoI which is covered by $\delta_t$, $\forall \delta_t \in \overline{D}$
- $\overline{D}_0 \subseteq O$, with $O$ as set of all MODDS
- $R(\overline{D}_0) \equiv \left\{x | x \text{ covered by } \delta_t, \forall \delta_t \in \overline{D}_0\right\}$
- Maximal sets of mutually overlapping detection discs: a maximal MODDS is not a subset of any other MODDS; i.e., $\overline{D}_0$ is maximal if $\forall x \in R(\overline{D}_0),\ \nexists\, \delta_t \notin \overline{D}_0$ covering $x$
- $M \subseteq D$: set of all maximal sets of mutually overlapping detection disc (MODD)
- $\tau_0$: set of targets detected by source at $x \in R(\overline{D}_0)$; if $\overline{D}_0 \in M$, then $\tau_0$ is the maximal set of targets detectable with a source at $x$

Using this Craparo and Karataş (2014) prove (see also Figure 2.1):

1. Possible optimal source locations $X$ can be found within intersection regions of all maximal sets of MODD, $X = \bigcup_{\overline{D}_0 \in M} R(\overline{D}_0)$.
2. $\forall \overline{D}_0 \in M$, $R(\overline{D}_0)$ contains either a target location, or an intersection point $i_{t,t'}$ between the boundaries of two detection discs.

Considering all of this, we can restrict our set of candidate positions for source placement to

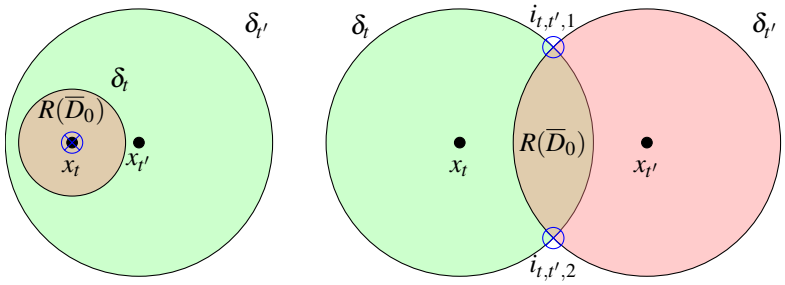


Figure 2.1: **Optimal Point Positions**—Candidate points $x_c$ for optimal point positions of sources lie within $R(\overline{D}_0)$. In case of no intersections one point inside the detection disc becomes a candidate position (left, e.g., $x_c = x_t$). When detection discs intersect, one of the intersection points $i_{t,t',j}$ becomes a candidate position (right, e.g. $x_c = i_{t,t',1}$).

the centers of DDs ($x_c = \{x_1, x_2, ..., x_T\}$) and the intersection points of DDs ($\bar{I} = \bigcup_{t,t' \in T} I_{t,t'}$). For the calculation of the intersection points, see Appendix A. Thus one can iterate over all centers and intersection points of all DDs to create the whole set of candidate locations ($C = \{x_1, x_2, ..., x_T\} \bigcup \bar{I}$) and determine for each of them the set of targets $\tau(c)$, which can be detected by the specific point. Then one can erase from this set all non-maximal subsets of targets as well as any duplicate sets to derive the reduced set of candidate locations. Craparo and Karataş (2014, p. 8) develop a five-step algorithm (LOC-GEN; see Figure 2.2) to do this preprocessing of data:

0. Input parameters are: set of receivers $R$ with positions $x_r, \forall r \in R$, set of targets $T$ with positions $x_t, \forall t \in T$, RoD $\rho_0$
1. Compute distances between targets and receivers $d_{t,r}, \forall t \in T, r \in R$, find $d_{t,r^*(t)} = \min_r d_{t,r}, \forall t \in T$, determine distances between all targets $d_{t,t'}, \forall t, t' \in T$
2. Create DDs with $\delta_t \in D$, center $x_t$ and radius $\rho_0^2 / d_{t,r^*(t)}, \forall t \in T$
3. For all pairs of DDs $\delta_t, \delta_{t'} \in D$ calculate all intersection points $I_{t,t'}$ and build the set of all intersections points $\bar{I} = \bigcup_{t,t' \in T} I_{t,t'}$, where each $I_{t,t'}$ can have between zero and two entries
4. Generate the set of all possible candidate location points $C = \{x_1, x_2, ..., x_T\} \bigcup \bar{I}$
5. Generate the reduced candidate point set by eliminating all points in $C$ which do not represent a set of MODDs

Figure 2.2: **Preprocessing to determine the set of all possible optimal source locations $C$ using algorithm LOC-GEN**

This final set of candidate locations is guaranteed to contain an optimal set of source locations. Craparo and Karataş develop two different approaches to select among these locations. One is an exact algorithm named OPT-LOC; Craparo and Karataş solve this integer

linear program (ILP) using general algebraic modeling system (GAMS) (see Figure 2.3). They also obtain a near-optimal solution based on a greedy algorithm (see Figure 2.4, GREEDY-LOC). The GAMS code can be found as a supplemental download at the Naval Postgraduate School (NPS) Dudley Knox Library.

**Indices and Sets:**

| | |
|---|---|
| $t \in T$ | targets, |
| $c \in C$ | candidate point locations, |
| $\tau(c)$ | set of targets detected by source at $c$. |

**Parameters:**

| | |
|---|---|
| $w_t$ | value of target $t$ |
| $k$ | number of available sources |

**Decision Variables:**

$$x_c = \begin{cases} 1 & \text{if source is placed at candidate location } c \\ 0 & \text{otherwise} \end{cases}$$

$$y_t = \begin{cases} 1 & \text{if target } t \text{ is detected} \\ 0 & \text{otherwise} \end{cases}$$

**Formulation:**

$$\max_{x,y} z = \sum_t w_t \cdot y_t \tag{2.1a}$$

Subject to

$$y_t \leq \sum_{c|t\in\tau(c)} x_c \qquad \forall t \in T \tag{2.1b}$$

$$\sum_c x_c \leq k \tag{2.1c}$$

$$0 \leq y_t \leq 1 \qquad \forall t \in T \tag{2.1d}$$

$$x_c \in \{0,1\} \qquad \forall c \in C \tag{2.1e}$$

Figure 2.3: Integer linear program OPT-LOC to determine the optimal source positions based on the set of possible optimal position $C$

## 2.3 Implementation of Preprocessing

The most complex algorithm in the optimization process is the preprocessing, realized by LOC-GEN. Craparo and Karataş (2014, p. 8) show that the algorithm has a complexity of $O(|T|^5)$. A realization of the preprocessing runs through to whole set of targets and

**Input Parameters:**

| | |
|---|---|
| $T$ | set of targets, |
| $C$ | set of candidate source locations, |
| $\tau(c)$ | set of targets detected by source at $c$, |
| $w_t$ | value of target $t$, |
| $k$ | number of available sources left to place. |

**Initialization:**

Initialize set of (nearly optimal) source locations $GL = \varnothing$

Initialize set of detected targets $GT = \varnothing$

**Greedy Selection:**

While $|GL| \leq k$, do:

$$GL \leftarrow GL \cup \operatorname*{argmax}_{c} \sum_{t \in GT \cup \tau(c)} w_t$$

$$GT \leftarrow \bigcup_{c \in GL} \tau(c)$$

Figure 2.4: **Greedy algorithm Greedy-LOC to determine the nearly optimal source positions based on the set of possible optimal position $C$**

determine all intersection points of the DDs. All intersection points and center points form the candidate list $C$. For each of these points the detectable targets $\tau(c)$ are calculated. Iterating through all $c, c' \in C$ s.t. $c \neq c'$ one can eliminate all locations $c$, which are not-maximal subsets, i.e., for which $\tau(c) \subseteq \tau(c')$.

An example of this procedure can be seen in Figure 2.5. Four targets $(x_i, i = 1, \ldots, 4)$ form eight intersection points of the detections discs $(i^k_{i,j}, i, j = 1..4, k = 1,2$, where $i$ and $j$ are targets and $k$ refers to the $k$th intersection point of the detection discs of targets $i$ and $j$). Therefore, $C$ initially contains 12 elements as possible source locations. One can now determine the set of detectable targets for each location. Iterating through all $c \in C$ leads to the detectable targets for each point as can be seen in Table 2.1. All center points of the targets are dominated, because they are all dominated by at least two of the intersection points. Thus, they can be eliminated from the final candidate location list. Points $\left\{i^1_{2,3}, i^1_{1,3}, i^2_{1,2}\right\}$ are non-dominated, while $\left\{i^2_{2,3}, i^2_{1,3}, i^1_{1,2}\right\}$ are dominated; thus the second set can also be eliminated. All elements in the first set can detect the same targets, which means they are equivalent for the optimization process. This makes it sufficient

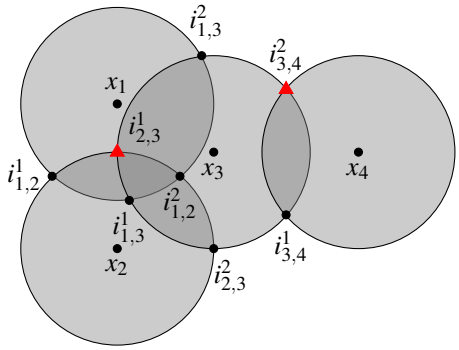


Figure 2.5: **Source Location Reduction**—All dominated points can be eliminated from the Candidate Point List $C$, therefore find all $c$ whose $\tau(c)$ are not-maximal subsets of targets. $\left\{i^1_{2,3}, i^1_{1,3}, i^2_{1,2}\right\}$ and $\left\{i^1_{3,4}, i^2_{3,4}\right\}$ are two sets of points representing maximal subsets of targets. These are non-dominated subsets. As possible location points for further analysis only $i^1_{2,3}$ and $i^2_{3,4}$ are chosen.

to take only one point out of this set into account for a possible source location. The second non-dominated set contains $\left\{i^1_{3,4}, i^2_{3,4}\right\}$. As in the first case, the elements enable the detection of the same targets, which is the reason that only only one point is considered in the following calculations. Thus to detect the four targets $x_i, i = 1,\ldots,4$, a set of two MODDs is necessary, represented by the points $i^1_{2,3}$ and $i^1_{3,4}$. This way the preprocessing algorithm leads to a reduction of the candidate point list from twelve to two entries in the final candidate location list.

Table 2.1: **Example of Dominated Location Elimination**—Starting with twelve possible locations, one can eliminate all center points as dominated locations. The locations colored in red are an example of non-dominated positions; an equivalent set would be $\left\{i^1_{1,3}, i^2_{3,4}\right\}$

| Center Point | Detected Targets | Intersection Point | Detected targets |
|---|---|---|---|
| $x_1$ | 1 | $i^1_{1,2}$ | 1,2 |
| $x_2$ | 2 | $i^1_{2,3}$ | 1,2,3 |
| $x_3$ | 3 | $i^1_{1,3}$ | 1,2,3 |
| $x_4$ | 4 | $i^2_{1,2}$ | 1,2,3 |
| | | $i^2_{1,3}$ | 1,3 |
| | | $i^2_{2,3}$ | 2,3 |
| | | $i^1_{3,4}$ | 3,4 |
| | | $i^2_{3,4}$ | 3,4 |

Craparo and Karataş (2014) use a direct implementation of the LOC-GEN algorithm from Figure 2.2 for the preprocessing. The intersection points and the set of all detectable targets for each of the entries in the set of possible candidate positions are calculated. Then the set of detectable targets for each of the candidate points is compared with the sets of detectable targets for all other candidate points. If a set is maximal, its candidate point is stored in the reduced candidate list. With increasing numbers of targets the computing time becomes large, because the algorithm does an exhaustive search of candidate locations. The complexity of $O(|T^5|)$ makes the algorithm ineffective for large numbers of targets, which is why Craparo and Karataş (2014, p. 15) state that a more efficient implementation of searching dominated sets should be developed.

## 2.4 Enhanced Preprocessing Algorithm

We now describe LOC-GEN-II, which is an adaption of LOC-GEN that performs this preprocessing more efficiently. LOC-GEN-II appears in Figure 2.6; the changes in comparison to Figure 2.2 are printed in blue. The full MATLAB code of LOC-GEN-II can be found in Appendix B as well as by supplemental download at the NPS Dudley Knox Library.

0. Input parameters are: receivers $R$, $x_r, \forall r \in R$; targets $T$, $x_t, \forall t \in T$, RoD $\rho_0$.
1. Compute $d_{t,r}, \forall t \in T, r \in R$, find $d_{t,r^*(t)} = \min_r d_{t,r}, \forall t \in T$, determine $d_{t,t'}, \forall t, t' \in T$.
2. Create DDs with $\delta_t \in D$, center $x_t$ and radius $\Delta_t = \rho_0^2 / d_{t,r^*(t)}, \forall t \in T$.
3. For all pairs of DDs $\delta_t, \delta_{t'} \in D$ calculate all $i^k_{t,t'}$ and build $\bar{I} = \bigcup_{t,t' \in T,k} \{i^k_{t,t'}\}$.
4. For each intersection point $i^k_{t,t'}$, determine the targets that can be detected from this location, $\tau(i^k_{t,t'})$.
5. Build the center point list $\overline{P} = \bigcup_{t \in T} \{x_t\}$.
6. If a target's DD has at least one intersection point, erase its center location from the list of center points $\overline{P}$.
7. Generate the final reduced candidate point list $\overline{C^*}$ by eliminating points in $\overline{P}$ and $\bar{I}$ which do not represent a set of MODDs by the following process:
   - Step 1. For each pair of center points $x_i$ and $x_j, i \neq j$, calculate the distance between them, $dist(i,j)$. If $dist(i,j) < \Delta_i$ and $dist(i,j) > \Delta_j$, erase $x_i$ from $\overline{P}$. Otherwise, leave $x_i$ in $\overline{P}$.
   - Step 2. Set $\overline{C^*} = \overline{P}$.
   - Step 3. While $\bar{I} \neq \emptyset$, find an intersection point that covers as many targets as possible. That is, find a point $i^k_{t,t'} \in \bar{I}$ such that $|\tau(i^k_{t,t'})| = \max_{i^k_{t'',t'''} \in \bar{I}} |\tau(i^k_{t'',t'''})|$. (Break ties arbitrarily.) Then, for each $i^k_{t'',t'''}$ such that $\tau(i^k_{t'',t'''}) \subseteq \tau(i^k_{t,t'})$, erase $i^k_{t'',t'''}$ from $\bar{I}$. Finally, add $i^k_{t,t'}$ to $\overline{C^*}$ and erase it from $\bar{I}$.
8. Return $\overline{C^*}$.

Figure 2.6: Enhanced preprocessing algorithm to determine the set of all possible optimal source locations $C$, **LOC-GEN-II**

There are two key differences between LOC-GEN and LOC-GEN-II. The first difference between the algorithms is that LOC-GEN-II discards a number of locations without comparison to other locations; this results in improved efficiency. A second difference is that LOC-GEN-II does not compare all possible pairs of locations; rather, it only compares locations of the same type (center point or intersection point).

The following lemma describes the insight that allows us to discard some locations without comparison to other locations:

**Lemma 2.1.** *For each target whose DD has at least one intersection point, the center point of the DD can be discarded as an optimal source location. The center point of a DD need only be included in the set of candidate source locations if the DD has no intersections.*

*Proof.* Consider a maximal set of mutually overlapping detection discs $\overline{D}_0 \in M$ that covers region $R(\overline{D}_0)$, and suppose the center point of a particular disc, $x_t$, lies within $R(\overline{D}_0)$. As noted by Craparo and Karataş (2014), the boundary of region $R(\overline{D}_0)$ consists of portions of the boundaries of detection discs $\delta_t \in \overline{D}_0$. Suppose the boundary of region $R(\overline{D}_0)$ consists of the boundary of a single detection disc, as in Figure 2.7 (left). In this case, $R(\overline{D}_0)$ contains no intersection points, and thus is must be represented by a center point in the set of possible optimal source locations. For the particular example in Figure 2.7 (left), $x_1$ is a final candidate point, while $x_2$ is dominated. Thus, the center of a disc with no intersections can, but need not be a final candidate point. Suppose now that the boundary consists of portions of the boundaries of multiple detection discs, as in Figure 2.7 (center). Craparo and Karataş (2014) prove that in this case there always exist intersection points $i^k_{1,2} \in R(\overline{D}_0)$, and thus $R(\overline{D}_0)$ need not be represented by a center point. For our particular example, $i^1_{1,2}$ and $i^2_{1,2}$ are each sufficient as candidate point, because each is an element of $R(\overline{D}_0)$, which is a MODD. As consequence, the centers $x_1$ and $x_2$ can always be discarded, even if one of them lies in $R(\overline{D}_0)$. Combining the two cases leads to a configuration as can be seen in Figure 2.7, right. The intersecting discs $\delta_2$ and $\delta_3$ form the common region $\overline{D}^{II}_0 \in M$ with $i^1_{2,3} \in R(\overline{D}^{II}_0)$ and with $i^2_{2,3} \in R(\overline{D}^{II}_0)$. As a consequence, $x_2$ and $x_3$ can be discarded, because $\overline{D}^{II}_0$ can be sufficiently characterized by one of the intersection points. But because $R(\overline{D}^{I}_0) \neq R(\overline{D}^{II}_0)$ and with $R(\overline{D}^{I}_0) = \delta_1$, this boundary is identical with the boundary of the disc $\delta_1$. The disc $\delta_1$ has no intersection and so the center $x_1$ must be an additional candidate point for this configuration.

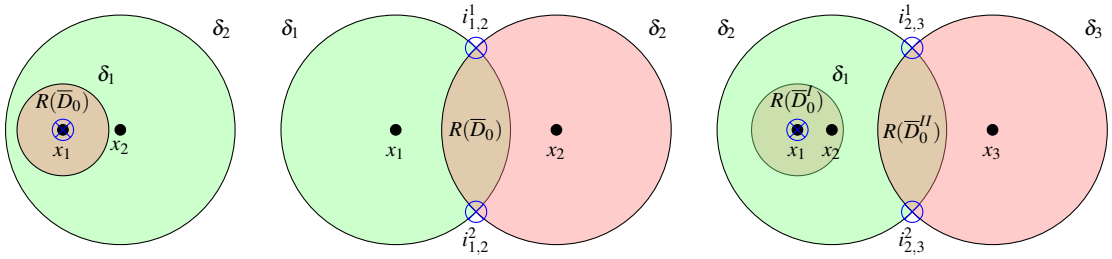


Figure 2.7: **Optimal Point Positions**—Candidate points $x_c$ for optimal point positions of sources lie within $R(\overline{D}_0)$.

□

Implementing this in the algorithm, at the beginning all targets without any intersections of their DDs are investigated. Let these form the set $\overline{P}$, a reduced set of target locations. Possible configurations of DDs without intersections can be seen in Figure 2.8 in the first two rows; the final possible candidate position is written above each of the graphics. The center points of non-overlapping DDs are all final possible candidate locations (Case IV). Overlapping DDs with the center of the larger disc not included in the area of the smaller disc always have the center point of the smaller DD as a final possible location (II, III). If both centers are part of both disc areas, either center point could be taken as a possible location (I). In the implementation we choose the center point of the larger DD. Combinations of overlapping DDs are possible and a mixture of the rules is used to determine the possible candidate location, as can be seen in Case V. $\{C1, C2\}$ and $\{C2, C3\}$ represent the case depicted in I, $\{C1, C3\}$ Case II. So a sensor at $x_{C2}$ is the final possible location. It enables the detection of all three targets. (Point $x_{C1}$ would also accomplish this.) For each element in the set $\overline{P}$, its affiliation to one of the Cases I to V is verified and the center point accordingly saved as a final candidate location into $\overline{C^*}$ or disregarded for further calculations.

In the last row of Figure 2.8, possible combinations of DDs with and without intersection points can be seen. Although these combinations could lead to a further reduction of the final candidate point list, these configurations are not incorporated in the algorithm. The two groups of DDs (with and without intersection points) are processed completely independently of each other. In Case VI it would be necessary to put $x_{C1}$ into the reduced list,

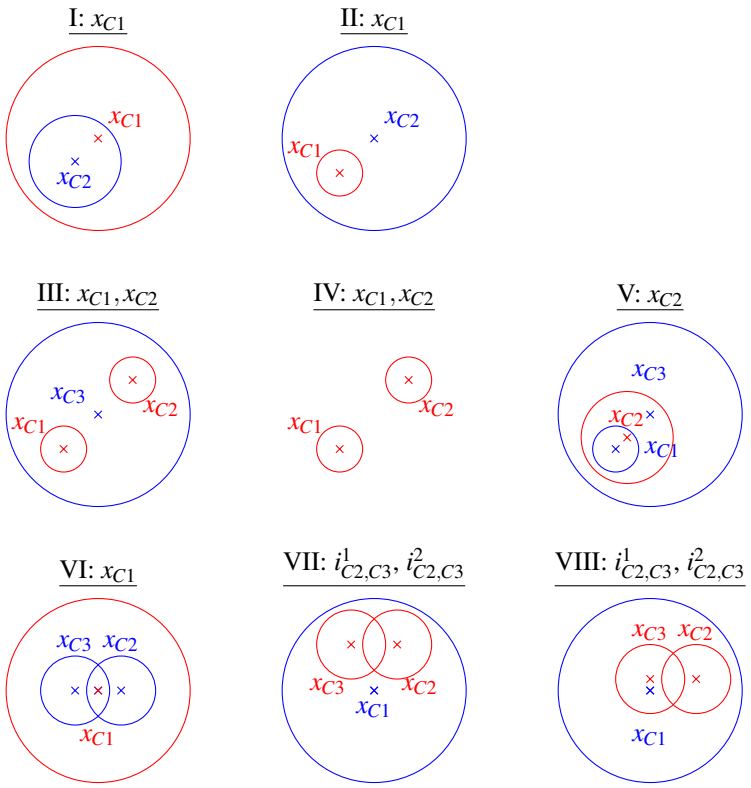


Figure 2.8: **Detection Discs with no Intersections**—Possible candidate points for optimal source location for different configurations of DDs without intersection. The non-dominated DDs are colored in RED and its center points are possible candidate locations. The dominated DDs are colored in BLUE; the center point of these discs can be neglected in further calculations.

in Cases VII and VIII one of the intersection points $i^1_{C2,C3}$, $i^2_{C2,C3}$. Yet, the computational effort to combine the processing of the two classes is comparably large. For example, for the group of non-intersecting discs, LOC-GEN-II only has to consider the distances to all other center points of DDs without intersections (amount $O(|T|)$. Whereas, incorporating the intersecting DDs would enlarge the work for each center point in the magnitude of $O(|T|^2)$, because then one has to assess the distances to all intersection points of all discs. Accordingly, computing time would grow quickly, which contradicts the aim of the new preprocessing algorithm to be lean and fast.

Fortunately, this modification does not sacrifice optimality: although we may add a dominated location to $\overline{C^*}$, we will never fail to include a non-dominated location. To see this, consider a location $x$. Assume without loss of generality that $x$ is an intersection point. If $x$

is dominated by another intersection point $y$, LOC-GEN-II will not add $x$ to $\overline{C^*}$. However, if $x$ is not dominated by any other intersection point but is dominated only by a center point, then LOC-GEN-II will add $x$ to $\overline{C^*}$. This is in contrast to LOC-GEN, which only adds non-dominated points to the final candidate point set. For example, in Case VI, LOC-GEN would put only $x_{C1}$ into the reduced list. Separating the two preprocessing parts determines $x_{C1}$ from non-intersecting DDs and one of the intersection points $i^1_{C2,C3}$, $i^2_{C2,C3}$ from the intersecting discs as possible locations. As a result two points are written in the list, both being equivalent with respect to the targets detectable from each. As a consequence, LOC-GEN-II leads to a possible enlarged set of reduced candidate locations without losing any important locations. However, this is accompanied by less computational complexity and shorter computing time in the preprocessing stage. Moreover, the follow-on algorithms for choosing among these locations have been shown to run quickly $O(|(\overline{C^*})||T|)$. So a slightly enlarged number of elements in the final candidate list $\overline{C^*}$ has as a consequence a much lower increase in computing time compared to the rise in complexity of combining the two preprocessing parts. Section 2.5 demonstrates this improvement computationally.

After having preprocessed targets whose DDs have no intersections, LOC-GEN-II considers the intersection points. For the greedy-like approach to find the rest of the final candidate locations in $\overline{C}^*$, LOC-GEN-II proceeds iteratively. As long as there is an element in the list of intersection points $\bar{I}$, LOC-GEN-II determines the element with the most detectable targets. Coming back to the example in Figure 2.5, this would be $i^1_{2,3}$ in the first iteration step. Then the algorithm stores this position in the final candidate location list $\overline{C^*}$ and searches through all other intersection points. If an intersection point is found which has the same or a subset of detectable targets, it is erased from $\bar{I}$. For the example in Figure 2.5 this would delete all points in the set $\left\{i^1_{1,2}, i^1_{1,3}, i^2_{1,2}, i^2_{1,3}, i^2_{2,3}\right\}$; thus after the first iteration step, there would only be $\bar{I} = \left\{i^1_{3,4}, i^2_{3,4}\right\}$ left. The algorithm repeats this step of searching for the largest set of detectable targets and eliminating all dominated intersection points until $\bar{I}$ is the empty set. Back to the example in Figure 2.5, in the second iteration step, point $i^1_{3,4}$ would be added to $\overline{C^*}$, then $i^1_{3,4}$ and $i^2_{3,4}$ would be eliminated from $\bar{I}$, because $\tau(i^2_{3,4}) \subseteq \tau(i^1_{3,4})$. As result, $\bar{I} = \emptyset$ leads to the termination of the algorithm in the second iteration step. This makes the algorithm very efficient, as can be seen in Section 2.5, because in each iteration step the number of elements which have to be investigated in the

following step is reduced drastically.

To determine an approximation of the order of LOC-GEN-II, one has to examine the two steps on their own. The number of targets without any intersection of its DD can never be larger than $|T|$, because it cannot exceed the number of targets. Step one of the algorithm has to compare each of these elements with one another, thus leading to a complexity of $O(|T|^2)$. The second step has to investigate all intersection points between the DDs, which can be at most $|T|^2$. So, looping through these intersection points and comparing their covered targets has a complexity of $O((|T|^2)^2 \cdot |T|) = O(|T|^5)$. Because the number of intersection points will normally be below $|T|^2$, this upper bound is a pessimistic one, as it is for LOC-GEN. Additionally, the number of elements, which have to be considered in the following step, are reduced by LOC-GEN-II in the preceding iteration step. Therefore the number of iteration steps can be bounded below with $|T| \cdot \log(|T|)$. The number reflects the complexity of algorithms that bisect the number of elements in each iteration step, such as Quick Sort or Merge Sort (Rosen, 2011, p. 367). LOC-GEN-II may not be able to divide the number of remaining elements in the following step in halves, but it is an estimation of the best performance the algorithm could reach. So the overall complexity of LOC-GEN-II will be in the magnitude of the following, where $|T| \cdot \log(|T|)$ has to be squared, because in each iteration step LOC-GEN-II has to compare the point with each entry remaining in the list

$$O(|T|^5) \geq O(\text{LOG-GEN-II}) \geq O(|T|^2 \cdot \log(|T|))$$

Further improvements occur due to accumulating the calculation of required intermediate results. For example, the event of finding an intersection point between two DDs does not only lead to the calculation and storage of those intersection points; it also results also in the direct elimination of the corresponding center points from the list of center points. Thus, calculations and different steps of the process are now merged, which reduces computing time.

An example of the result of the whole process can be seen in Figure 2.9. On the left side, the given set of targets $t_i, i = 1..10, (\times)$ and receivers $r_j, j = 1..10, (\blacktriangle)$ are displayed. In preprocessing, first the candidate point list is generated (right side, $\square$) and reduced by all targets without intersecting DDs (here none) and all dominated intersection points. This

results in the final list of candidate positions $\overline{C^*}$ (left, $c_k, k = 1,..5,$ ●, and right, ×). OPT-LOC or GREEDY-LOC is then used to determine optimal locations for a given number of sources (here two sources, right side, ○). As result, the two sources would enable the detection of the seven targets $x_2, x_4, x_5, x_6, x_7, x_8, x_9$.

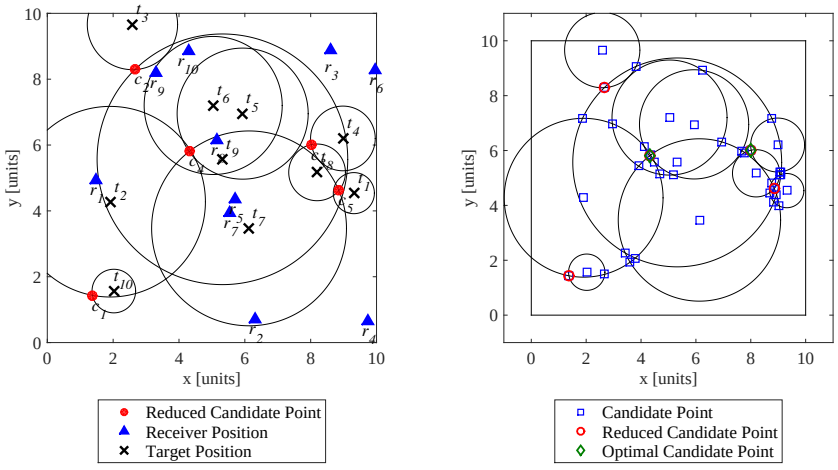


Figure 2.9: **Determination of Optimal Source Positions**—Starting with a set of targets and receivers (left) the preprocessing delivers the list of all possible source location candidates (right), which are reduced to the final set of candidate locationsguaranteed to contain the optimal source locations.

## 2.5 Computational Comparison of LOC-GEN and LOC-GEN-II

We now describe the results of different simulation experiments comparing LOC-GEN and LOC-GEN-II. We use the following computer configuration: AMD K16 Mullins Processor (A4-6210) with four kernels and 1800MHz clock frequency per kernel, 100MHz BUS clock frequency, operating system Microsoft® Windows 8.1 64-Bit, 6144 MBytes DDR3 RAM with clock frequency 798.4 MHz. Software was MATLAB® R2014b, version 8.4.0.150421. We consider the following scenario: A square area of size $10 \times 10$ units has to be investigated. It contains $m$ targets and $k$ receivers, which are all randomly placed. The RoD is set to $\rho_0$. The task is to find the final reduced set of candidate source locations $\overline{C^*}$.

We compare LOC-GEN-II and LOC-GEN in order to find out whether they deliver equivalent results for candidate locations. Positions of targets and receivers for both algorithms are the same for each instance. In 50 trials, all experiments lead to identical numbers of final candidate locations in $\overline{C^*}$, with the same targets detectable by equivalent candidate locations. Two examples represent these results. The first configuration ($m = k = 15, \rho_0 = 1.0$; see Figure 2.10) has as result a set of eleven candidate locations in $\overline{C^*}$. The starting points of LOC-GEN are 33 possible locations; for LOC-GEN-II the number is reduced to 25, because the DDs of eight targets have intersections and thus their center points are discarded. The computing times are 0.1250 seconds for LOC-GEN-II and 0.3281 seconds for LOC-GEN. In the second example ($m = 150, k = 50, \rho_0 = 0.6$; see Figure 2.11), it becomes clear

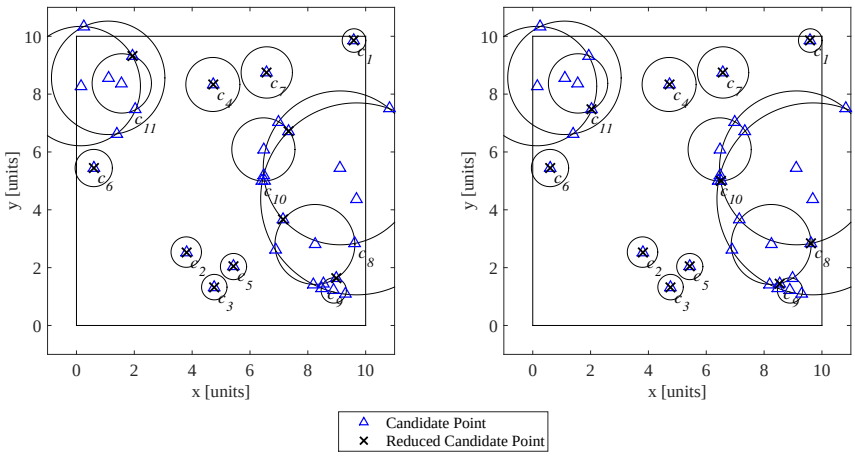


Figure 2.10: **Final Candidate Locations for 15 Targets and 15 Receivers**—left: LOC-GEN-II, right: LOC-GEN. Although some candidate locations do not coincide, they do represent the same targets detectable by them. For example, $c_{11}$ of LOC-GEN was not chosen by LOC-GEN-II, but the equivalent location at $(x, y) \approx (2, 9)$. Both can detect the three targets in the upper left corner.

that separation in dominated and non-dominated candidate locations becomes challenging and computationally not trivial. Both algorithms end up with a set $\overline{C^*}$ with 140 entries, where LOC-GEN started with 1,512 possible locations and LOC-GEN-II with 1,362. In this case of a very high number of DDs in a relatively small area, all DDs have at least one intersection. Because of that, no center points of the discs are taken into account by

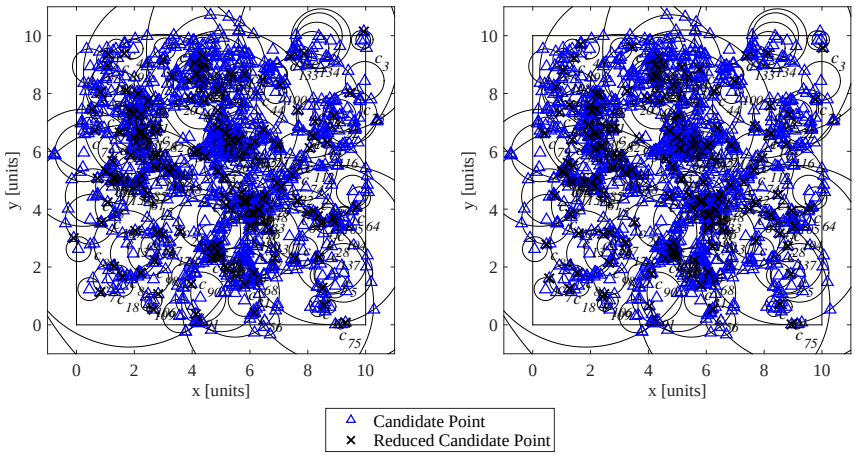


Figure 2.11: **Final Candidate Point Location for 150 Targets and 50 Receivers**—left: LOC-GEN-II, right: LOC-GEN. Example of growth of candidate position number with rising number of targets and receivers.

LOG-GEN-II. The computing times are 3.5014 seconds (LOC-GEN-II) and 5.1623 seconds (LOC-GEN). The main reason for the lower computing time of LOC-GEN-II is the successive reduction of the number of remaining candidate points. Our implementation of LOC-GEN takes each candidate point and for it loops through the whole list of candidate points $C$. So it always needs $|C|^2$ steps. LOC-GEN-II reduces in each iteration step the list of remaining candidate locations by eliminating dominated or equivalent locations. The number of points, which have to be taken into account in the following step, decreases sometimes drastically, until no elements are left in the $\bar{I}$. As consequence, LOC-GEN-II never loops through all elements in $C$ but through a greatly decreased number of elements, as can be seen in Figure 2.12. Note that this elimination could also easily be incorporated into LOC-GEN; however, LOC-GEN still compares center points and intersection points, while LOC-GEN-II considers these two classes of points separately.

The reduction of entries in the candidate point list per iteration step leads to a termination of the LOG-GEN-II in case one after four steps, in case two after 140 steps. These are the iterations based on the intersection points of DDs, and so of the second part of LOG-GEN-II. It must be considered that LOC-GEN-II also analyzes the targets whose DD have

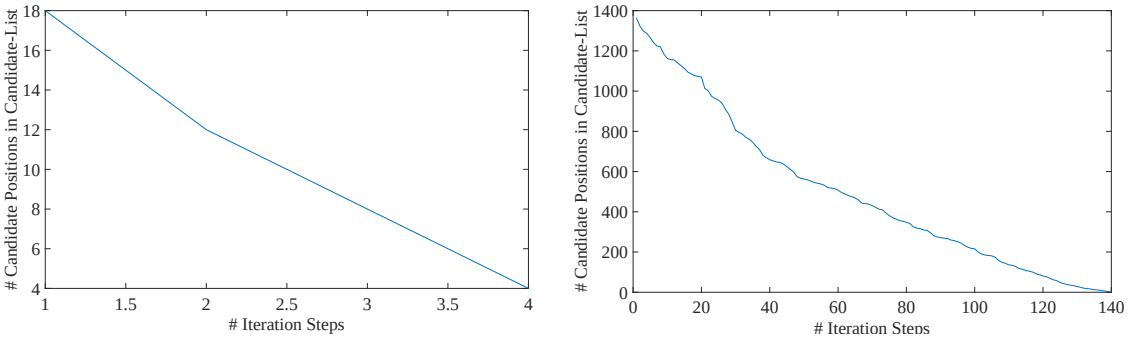


Figure 2.12: **Iterative Reduction of Elements in Candidate Point List**—Number of remaining candidate points in each iterations step using LOC-GEN-II algorithm. Left side: $m = k = 15, \rho_0 = 1.0$; right side: $m = 150, k = 50, \rho_0 = 0.6$. The great decrease of remaining elements has as effect the finalization of the preprocessing in essentially fewer iterations steps, compared to LOC-GEN, which always needs $|C|$ iteration steps in our implementation

no intersections (the first part of LOG-GEN-II). This adds for the first case seven steps to the algorithm. In contrast, LOC-GEN loops through all pairs of elements of the starting set $C$, which includes all intersection points and all center points (33 in case one and 1,512 in case two). The right graph of Figure 2.12 illustrates the effective reduction of remaining candidate points. Using LOC-GEN-II to find all possible optimal locations, the calculation aborts after 140 iteration steps, which is less than one tenth of the 1,512 steps LOG-GEN needs. This is the main reason for the significantly lower computing times of LOC-GEN-II.

To prove these results with statistically significant data, we conduct two main experiments with the common elements of AoI of size $(10 \times 10)$ units, RoD of $\rho_0 = 0.6$ and the following settings:

1. $m = 5, 10, 15, \ldots, 125$ targets and $k = 25, 50, 75, 100$ receivers are placed randomly in the AoI. For each of these configurations we measure the time for preprocessing using LOC-GEN. We replicate this experiment $2 \times 150$ times. We repeat this experiment using LOC-GEN-II. For all replications the placement of targets and receivers are chosen randomly.
2. $m = 50, 100, 150, \ldots, 500$ targets and $k = 50$ receivers are placed randomly in the AoI. For each of these configurations we measure the time for preprocessing using LOC-GEN. We replicate the experiment 100 times. We repeat this experiment using LOC-GEN-II. For all replications the placement of targets and receivers are chosen

randomly.

Besides computing time, we also store the number of candidate locations at beginning and end of each process. We discuss these results in section 2.7.

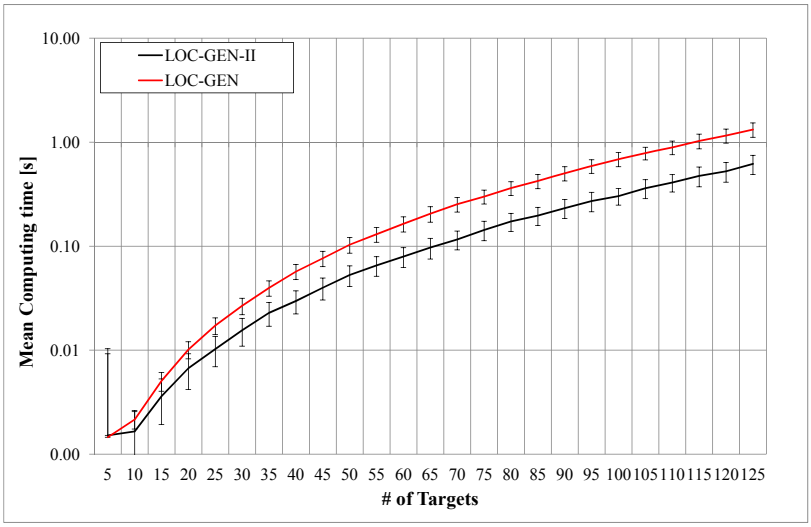


Figure 2.13: **Candidate Point Reduction—Comparison of Algorithms—25 Receivers—** Preprocessing time to compute the final set of candidate locations for different numbers of targets and 25 receivers, RoD $\rho_0 = 0.6$, using LOC-GEN and LOC-GEN-II.

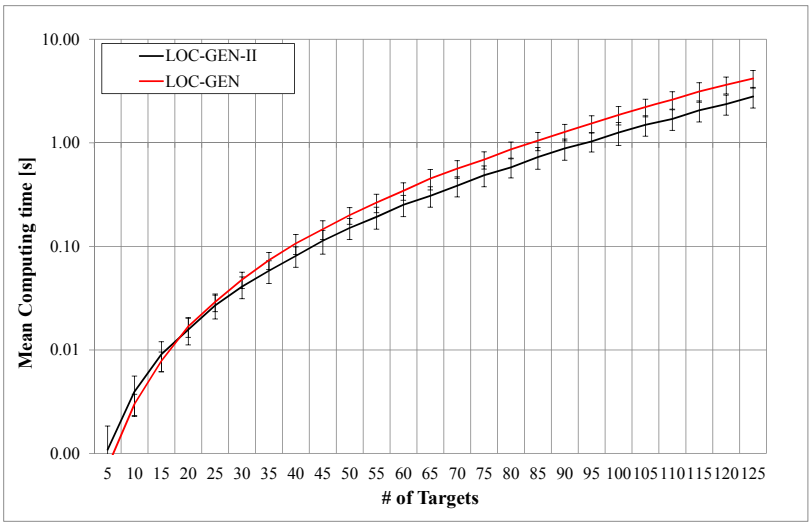


Figure 2.14: **Candidate Point Reduction—Comparison of Algorithms—100 Receivers—** Preprocessing time to compute the final set of candidate locations for different numbers of targets and 100 receivers, RoD $\rho_0 = 0.6$, using LOC-GEN and LOC-GEN-II.

The results of experiment one for 25 and 100 receivers can be seen in Figure 2.13 and Figure 2.14, and for experiment two in Figure 2.16. All graphs have as common elements the significant difference between computing times of both algorithms and an increase of computing time with rising number of targets. This growth can be observed in the logarithmic-scaled plot as an approximately straight line for a large number of targets. For a rising numbers of targets LOC-GEN-II shows a lower increase in computing time compared to LOC-GEN. In the graphs also error bars are depicted, which represent the standard deviation of the runs for each design point (see Figure 2.13). Especially for a large number of targets, the reduction in computing time using LOC-GEN-II becomes obvious. There can also be configurations of higher numbers of receivers and very small numbers of targets, where LOC-GEN-II does not perform better than LOC-GEN. However, in these cases the overall computing time is comparably small. With an increasing number of targets (above $|T| > 50$) and thus an increasing computing time, LOC-GEN-II outperforms LOC-GEN significantly. This can be explained by the fact that executing a computer program consists of two main parts: administrative procedures of the programming language and the task-based computing time. Examples of the first kind are memory allocation, variable implementation and so forth. The task itself means the execution of mathematical operations like finding intersection points, looping through lists, and so forth. The above mentioned configuration (small number of targets, relatively higher number of receivers) results in only very few possible candidate locations (e.g., 10 to 20 possible locations in the overall candidate list $C$). With this, MATLAB spends the main part of computing time for the administrative part, whereas the pure mathematical computations use only a small amount of the overall time. As a consequence, both algorithms need almost the same computing time, because the administrative part requires an equal amount of time for each. This explains also the large variations in these results, depicted by the error bars. When the number of targets becomes large compared to the number of receivers, the mathematical part of the computation overtakes the administrative part, leading to better performance of LOC-GEN-II. In terms of absolute values of computing time and for $|T| = 125$, the following results can be observed:

- $|R| = 25$: computing time for LOC-GEN$= 1.32 \pm 0.212$ seconds
  computing time for LOC-GEN-II$= 0.619 \pm 0.130$ seconds
  LOC-GEN-II is on average 53% faster

- $|R| = 100$: computing time for LOC-GEN$= 4.18 \pm 0.807$ seconds
  computing time for LOC-GEN-II$= 2.79 \pm 0.631$ seconds
  LOC-GEN-II is on average 33% faster

The large standard deviations of different runs for one set of parameters can be explained by chosen random locations for receivers and targets for each run. Figure 2.15 shows all preprocessing times for experiment one using LOC-GEN-II algorithm. With an increasing number of targets as well as receivers, computing time grows, but the impact of $|R|$ is much less important. Although a doubling of the $|T|$ has as an effect an approximately multiplying tenfold of the computing time, doubling $|R|$ results in nearly doubled computing time. Thus, number of targets is the driving parameter for preprocessing time. When one also

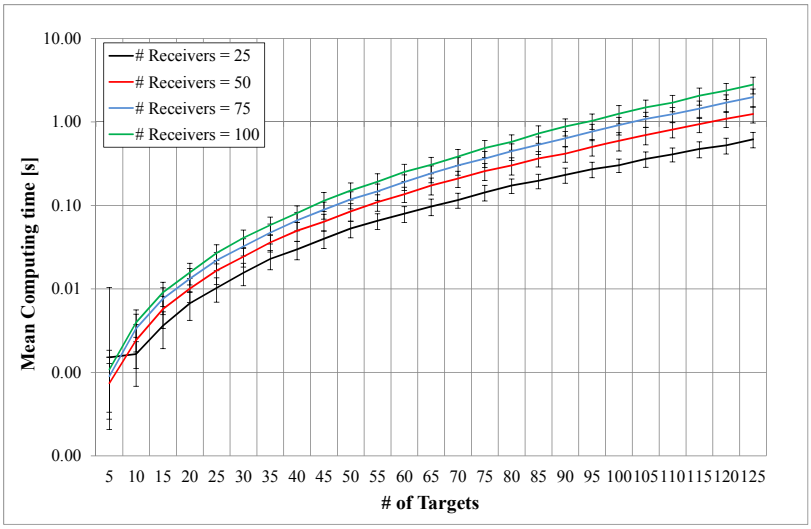


Figure 2.15: **Candidate Point Reduction—Computing Time—LOC-GEN-II**—Preprocessing time to compute the final set of candidate locations for different numbers of targets and receivers, RoD $\rho_0 = 0.6$, using LOC-GEN-II.

takes into account the results of experiment two, the efficiency of LOC-GEN-II becomes even more apparent. The large number of targets in the AoI have, as a consequence, a large number of intersection points and also possible source locations. Thus, an algorithm that reduces this number of candidate points in each step has a greater advantage compared to an algorithm that has to cycle through all intersection points. As a result, LOC-GEN-II needs on average less computing time. For more than 100 targets the differences exceed the sum of the standard deviations of both algorithms and they grow further with a rising number of

targets (see Figure 2.16). To put it in numbers, LOC-GEN-II starts with 40% lower com-

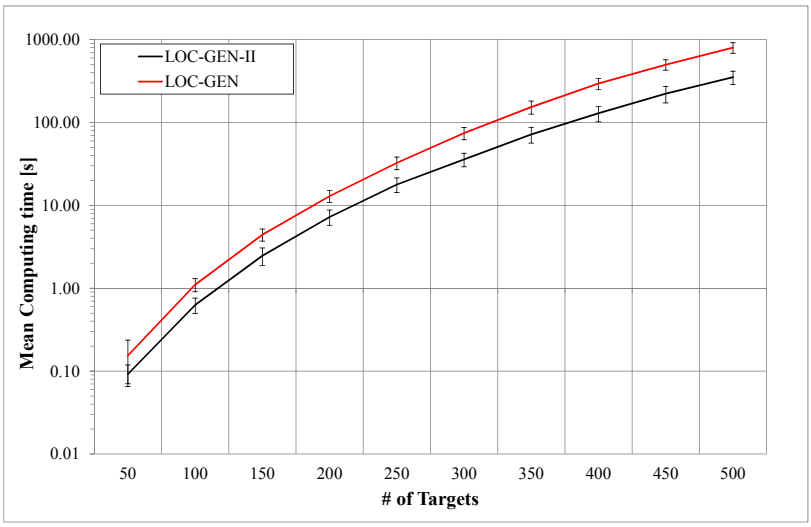


Figure 2.16: **Candidate Point Reduction—Comparison of Algorithms—Large Number of Targets**—Preprocessing time to compute the final set of candidate locations for experiment two with large number of targets and 50 receivers, RoD $\rho_0 = 0.6$, using LOC-GEN and LOC-GEN-II.

puting time than LOC-GEN for 50 targets, is around 45% faster for 100 up to 250 targets, and reaches 56% better performance for 500 targets. LOC-GEN needs $800.21 \pm 116.966$ seconds to preprocess a set of 500 targets and 50 receivers, where LOC-GEN-II requires $351.84 \pm 64.003$ seconds. Moreover, the one-standard deviation error bars do not, overlap for most of the problem sizes, considered.

To summarize, LOC-GEN-II achieves significantly better computational performance compared to the LOC-GEN algorithm, while delivering an equivalent final set of possible candidate locations for sources. It leads to a decrease of computing time between 30% up to more than 50% with statistical significance of the results, proven by several replications of different design points.

## 2.6 Connection of Preprocessing and Optimal Source Location

We now investigate the whole process of finding optimal source locations, and specifically overall computing time required. Therefore, we implement LOC-GEN-II as preprocessing

algorithm and OPT-LOC (see Figure 2.3) and GREEDY-LOC (see Figure 2.4) to find optimal source positions. To solve the OPT-LOC ILP, we use GAMS Release 24.2.1 r43572 for x86_64/MS Windows; for all other calculations we use MATLAB in the configuration of Section 2.5. The question is whether it is more efficient to terminate LOC-GEN-II early and begin with a larger set of candidate locations. The reason for these thoughts is the lower complexity of GREEDY-LOC ($O = (|\overline{C^*}|^2)$) compared to LOC-GEN-II ($O(|T|^5)$). We develop the following procedure:

- $m = 20, 40, .., 200$ targets and $k = 50$ receivers with $\rho_0 = 0.6$ are placed randomly in the AoI, and $l = 10, 20, \ldots, 50$ sources can be optimally placed. For each design point, 100 replications are computed.
- Preprocessing starts with LOC-GEN-II until one of the following criteria is fulfilled:
  1. $x = 0.1, 0.125, \ldots, 0.4\%$ of the starting number of candidate points $|\overline{P} \bigcup \bar{I}|$ are found to be final candidate locations, or
  2. $10 \cdot l$ final candidate locations are found by LOC-GEN-II
- then the remaining elements of $\overline{P} \bigcup \bar{I}$ are added to $\overline{C^*}$
- and OPT-LOC and GREEDY-LOC are used to determine optimal source locations.

The thresholds $x = 0.1, 0.125, .., 0.4\%$ were determined experimentally to be the magnitude where a significant reduction of preprocessing work could be eliminated by an early termination of LOC-GEN-II compared to processing the whole set. The second threshold $10 \cdot l$ has its root in the fact that LOC-GEN-II uses a greedy-like approach. The possible candidate locations are put in descending order of their ability to detect targets in the final candidate set, starting with the most promising locations. Although the targets are assigned weights $w_t$ in the optimization process, we find that the probability that a final optimal source location will not be in the $10 \cdot l$ best results from LOC-GEN-II is insignificantly low. With these criteria a sharing between the burden between complex preprocessing and substantially faster optimization processes is realized.

The results of this experiment can be seen in Figure 2.17, where the computing times for different parts of the process are depicted for the cases of a present or absent break in preprocessing. As expected, preprocessing time can be reduced using an earlier abortion, but way computing time of the optimization process increases. On the other hand, when there is no break in preprocessing we see a larger preprocessing but shorter optimization

time. Moreover, the largest reductions can be reached for small numbers of targets ($|T| \leq 60$), where preprocessing time itself is small. Overall, the absolute gain from the earlier termination is not large. Computing time for Greedy-LOC is abbout an order of magnitude below the two other processes and shows only a slight dependency on target number and abortion of preprocessing, therefore it can be neglected.

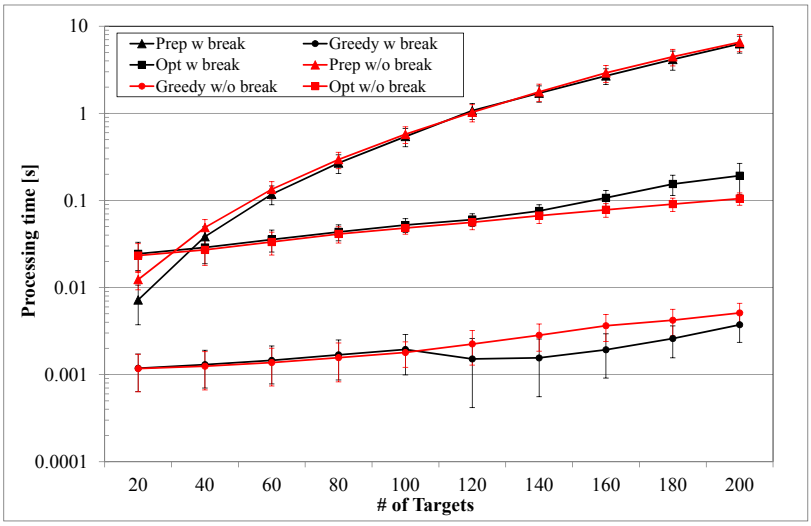


Figure 2.17: **Reduction of Preprocessing and Optimization Time—Overview**—The computing time for preprocessing and optimization for different number of targets and complete or aborted preprocessing.

The resulting summation of preprocessing and optimization computing times can be seen in Figure 2.18. Here it becomes obvious that the effect of a breakdown of preprocessing does not lead to a significant reduction of the overall computing time. Although the reduction reaches 10% to 20% of the overall computing time, the different runs show so many variations (reflected by overlapping one-standard deviation error bars), making a distinction between the two different preprocessing ways is not reasonable.

This leads to the result that abortion of preprocessing has an effect on the overall computing time, but this effect shows large variations and the increase of performance is not as high as expected. Therefore, the disadvantages of a reduced preprocessing algorithm (incorporation of break criteria in preprocessing is computationally challenging and have as outcome suboptimal preprocessing result) outweigh the slight advantages in computing time, which leads to the decision to discard this possibility of further reduction of computing time in

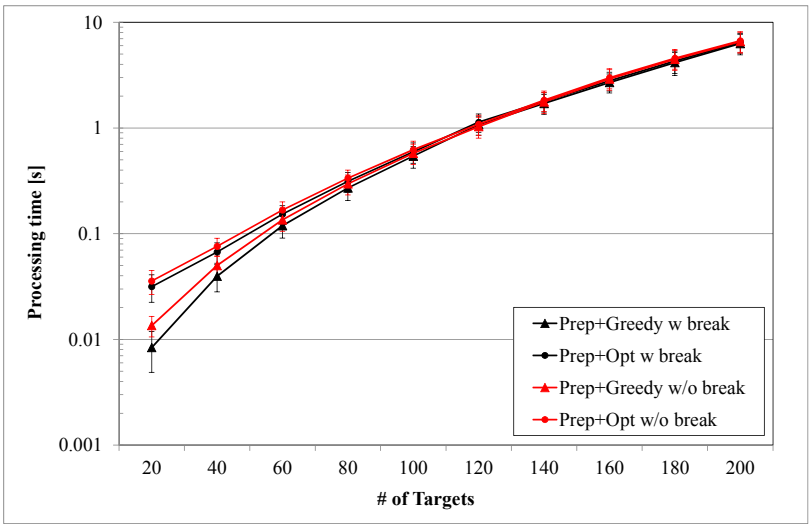


Figure 2.18: **Reduction of Sum of Preprocessing and Optimization Time—Overview—** The sum of the computing time for preprocessing and optimization for different number of targets and complete or aborted preprocessing.

this work.

## 2.7 Investigation of Number of Elements in the Candidate Point Lists

Craparo and Karataş (2014, p. 15) show an increase of candidate locations with the number of targets for a fixed number of 50 receivers before and after preprocessing. The increase before preprocessing is obvious, because enlarging the number of targets results in higher numbers of center points as well as intersections between DDs and in more candidate locations. But the simulation shows an approximately an exponential gain in the number of candidate locations after preprocessing. This increase is not obvious and was not explained in the study; for this reason we now take a closer look at these observations. Analyzing the number of elements in $\overline{C^*}$ (see experiment one, Section 2.5) leads to Figure 2.19. The numbers coincide with the observations Craparo and Karataş make in their simulation for $k = 50$ receivers and different numbers of targets.

By introducing additional variation in the number of receivers, an interesting observation can be made. Between 25 and 65 targets a shift in the slope of the curves can be observed. For low $|T|$ the configuration with the smallest $|R|$ leads to the highest $|\overline{C^*}|$. However, high

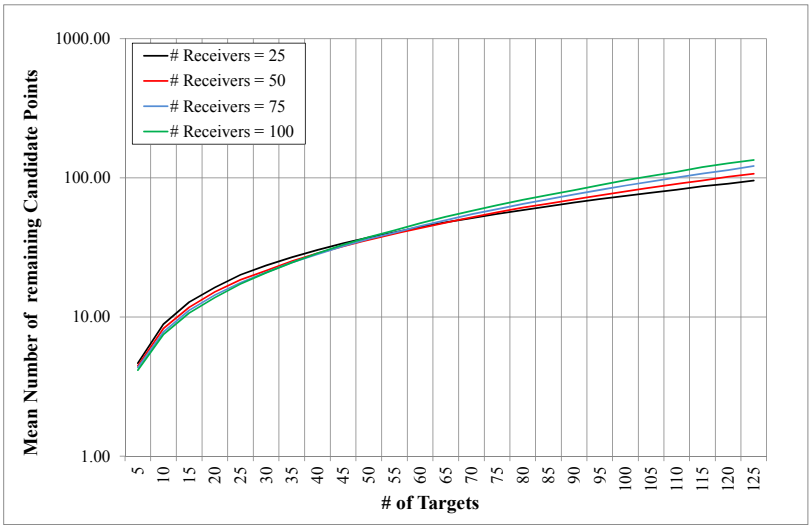


Figure 2.19: **Number of Elements in the Final Candidate Location List**—Number of candidate points in the final list after preprocessing for $m = 5,\dots 125$ targets and $k = 25,\dots 100$ receivers.

$|T|$ and small $|R|$ result in the lowest $|\overline{C^*}|$. To analyze this problem, two phenomena are addressed in this section, based on new simulation setups.

1. How can it be explained that for different numbers of receivers and rising number of targets the increase in the number of elements in $\overline{C^*}$ changes?
2. How is the number of elements in the final possible location list $\overline{C^*}$ related to the number of targets and receivers?

For a set of $m = 5,10,\dots,100$ targets and $k = 20,40,\dots,100$ receivers, all randomly positioned in an AoI of $10 \times 10$ units, and $\rho_0 = 0.6$, LOC-GEN-II is used to generate the final candidate location list. For each configuration 100 replications are made. The overall number of elements in $\overline{C^*}$ is analyzed, as well as the number of elements in $\overline{C^*}$ generated by center points (CPs) or by intersection points (IPs) of DDs. The results appear in Figure 2.20 and Figure 2.21; error bars in the first plot illustrate the standard deviation between different replications. For up to 50 targets the increase of candidate locations with rising number of targets is subexponential; for more targets it flattens to nearly exponential. This can be observed by the curves, having in the logarithmic scale a clear concave shape up to 50 targets and then converging to a constant slope. Relative variations between different replications become smaller with a higher number of targets, making a distinct discrimination

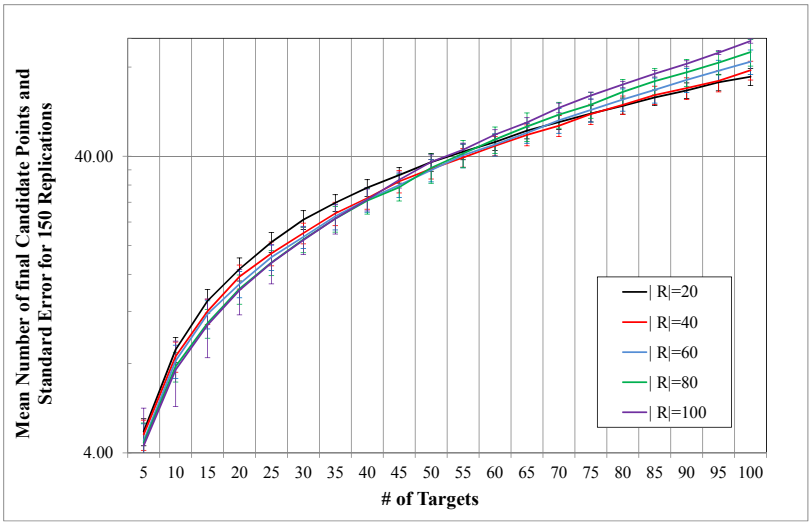


Figure 2.20: **Final Candidate Points Distribution**—Number of candidate points in the final list after preprocessing for different number of targets and receivers.

between results for different receiver numbers possible. The reason for this phenomenon can be seen in Figure 2.21 and is based on the composition of the final candidate location list. Whether an element in $\overline{C^*}$ is a CP or an IP of DDs has the main influence on the increase.

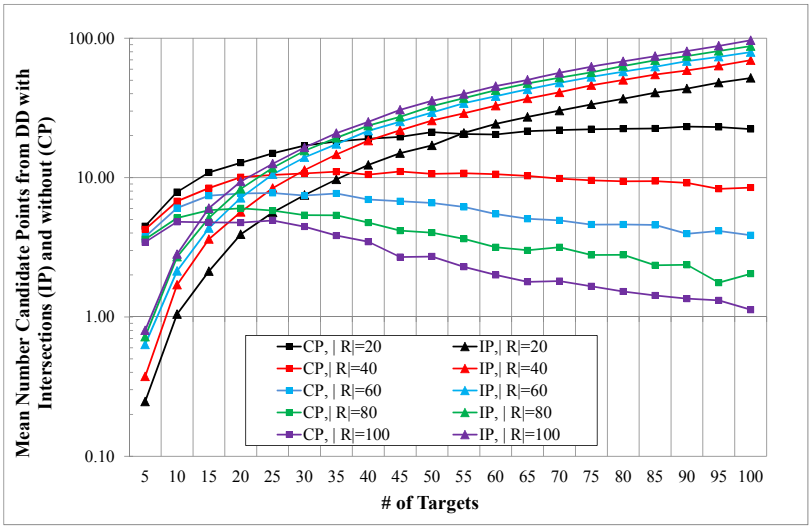


Figure 2.21: **Final Candidate Points Distribution**—Number of candidate points from CPs and IPs after preprocessing for different number of targets and receivers.

These two groups of points follow an opposing trend. A small number of targets is related to few IPs; many of the DDs even do not overlap. So target positions themselves are very often the only possible location to detect the corresponding target. As consequence, CPs are dominating elements in the final candidate location list, when target numbers are low. With increasing $|T|$, the number of center points as possible candidate locations can only grow with $O(|T|)$, whereas the number of intersection points grow with $O(|T|^2)$. Moreover, a DD with at least one IP can never have the CP as a possible candidate location, as shown in Lemma 2.1. As a consequence, a rising number of intersections leads to a corresponding decrease in CPs as candidate locations. This behavior is clearly observable in Figure 2.21 with an exponential increase of IPs and a stagnation or decrease of CPs when target numbers grow.

Superimposed, one can observe the influence of the receiver numbers on the curves and the composition of CPs and IPs. A configuration with a large number of receivers implies a high density of them. This results in a high probability that a receiver is closer to a target location and so the minimal distance between target and receiver $d_{t,r^*}$ becomes smaller. But this means also larger radii of the DDs, which leads to a higher possibility of overlapping and intersecting each other in an AoI with constant size. As a consequence, the number of IPs as candidate locations as well as final candidate positions rises. On the other hand, more DDs have intersections, so their CPs are automatically erased from the candidate point list, leading to fewer CPs in the final candidate list. When, additionally, the target number becomes larger, the trend of more IPs and less non-intersected DDs is intensified, what results in an accelerated decrease of CPs, as can be seen in the violet and green curves, especially.

On the other hand, very few receivers and thus a lower density of them enlarge the smallest distance to targets on average. The possibility of smaller DDs increases, leading to fewer IPs. Relatively more DDs will have no intersections; hence the number of CPs in the final list is comparably large also for a higher number of targets (see especially black and red lines in Figure 2.21). For 20 receivers the number of CPs has a nearly constant level of 20 for $30 \leq |T| \leq 100$. Having 100 receivers instead, the magnitude drops down from four to one for the same range of target numbers.

Putting these results together, the number of CPs has the most significant influence on the

number of candidate location points for small target numbers. For larger values of $|T|$ the number of IPs has the most influence. Additionally, the number of receivers influences the composition of CPs and IPs substantially. This leads to the effect shown in Figure 2.20, in which a flipping of the curves for different numbers of receivers with an increasing number of targets can be observed.

For the above mentioned experiment, the following relative changes in the number of candidate points can be observed when doubling the number of targets (see Table 2.2). They show clearly the overall trend of increasing elements in the final candidate location list with rising numbers of target sets as well as the influence of varying receiver sets. Equivalent

Table 2.2: Increase in Number of Candidate Points with Rising Target Number for Different Numbers of Receivers

| **targets** | **number of receivers** | | | | |
|---|---|---|---|---|---|
| **from - to** | **20** | **40** | **60** | **80** | **100** |
| $5-10$ | 0.8898 | 0.8341 | 0.8619 | 0.8105 | 0.8003 |
| $10-20$ | 0.8700 | 0.8560 | 0.8142 | 0.8255 | 0.8524 |
| $15-30$ | 0.8772 | 0.8346 | 0.8218 | 0.9130 | 0.9351 |
| $20-40$ | 0.8781 | 0.8376 | 0.9286 | 0.9818 | 1.0217 |
| $25-50$ | 0.8617 | 0.9264 | 0.9635 | 1.0875 | 1.1886 |
| $30-60$ | 0.8263 | 0.9688 | 1.0483 | 1.1730 | 1.2652 |
| $35-70$ | 0.8697 | 0.9761 | 1.1084 | 1.2426 | 1.3616 |
| $40-80$ | 0.8885 | 1.0630 | 1.1710 | 1.3258 | 1.4424 |
| $45-90$ | 0.9284 | 1.0620 | 1.2642 | 1.4423 | 1.4625 |
| $50-100$ | 0.9393 | 1.1496 | 1.3229 | 1.4559 | 1.5556 |

numbers can also be found in the data of experiment one in section 2.5. The reason for this is the same magnitude of the parameters in both experiments. For experiment two of Section 2.5 the range of increase in number of candidate points goes from 1.245 (doubling from 50 to 100 targets) up to 2.167 (doubling from 250 to 500 targets). To determine a model for the number of final candidate points as a function of receiver and target number, we construct a regression model based on the simulation data. We use the JMP software to find the best model using the Minimum BIC stopping rule and considering the variables up to the power of three and their interactions. This setup captures the expected increase of

candidate locations with increasing target numbers. The result is shown in Equation 2.2:

$$x_{FCP} = -19.4983 + 0.9178 \cdot m + 0.1350 \cdot k + 0.0029 \cdot (m - 69.0476)^2 + 0.0011 \cdot (k - 60.7143)^2 + 0.0041 \cdot (m - 69.0476)(k - 60.7143) \quad (2.2)$$

where $m$ is the number of targets, $k$ the number of receivers, and $x_{FCP}$ the fitted number of candidate points. The corresponding output of JMP can be seen in Figure 2.22. All parameters are significant at an extremely high level and the overall performance of the fit is with a value of $R^2_{adj} = 0.999664$, exceedingly high. The strong dependency between

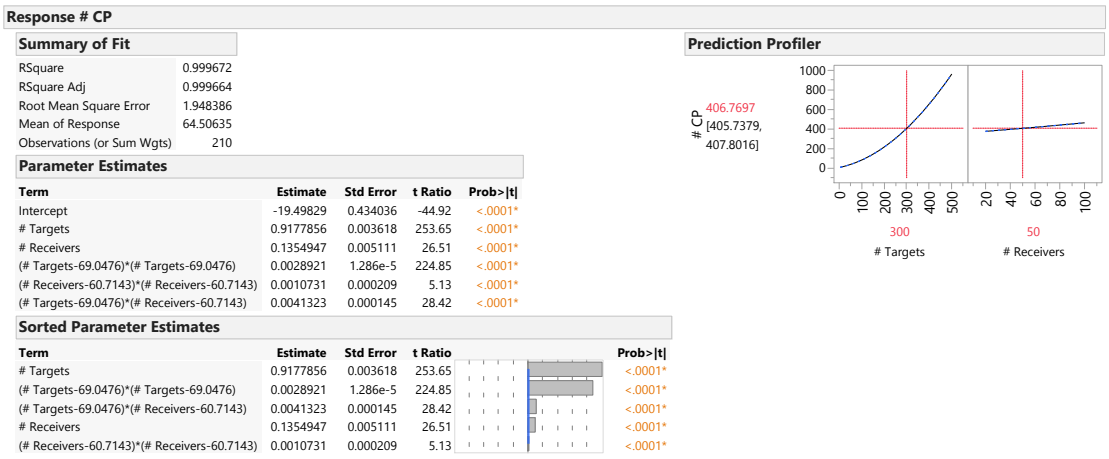


Figure 2.22: **Regression Fit for Number of Candidate Points as Function of Number of Targets and Receivers**—Result of JMP model regression fit using targets and receivers numbers as input (linear, quadratic and cross terms).

the number of candidate points and number of targets can be seen in the relatively large coefficients for these parameters and the prediction profiler plot; the weaker dependency to the number of receivers shows especially in the prediction profiler plot. The results underline the observations that the target number has the main impact on the number of final candidate points, whereas the receiver number plays only an inferior role. Comparing the fitted results with the simulation data, especially for a low number of receivers and targets, one can observe larger differences and the fitted values can lie outside the one standard deviation error bars (see Figure 2.23). But for the values $|T| > 20$ of $|R| >> 20$ the fir lies within these error bars. With increasing $|T|$, the predicted number of candidate points using the regression fit converges very quickly to the simulation results. For more

than 40 targets the difference between the two curves is negligible and in some parts the curves even overlap, as can be seen for $|T| > 50$ and $|R| = 60$. So, for the given AoI of

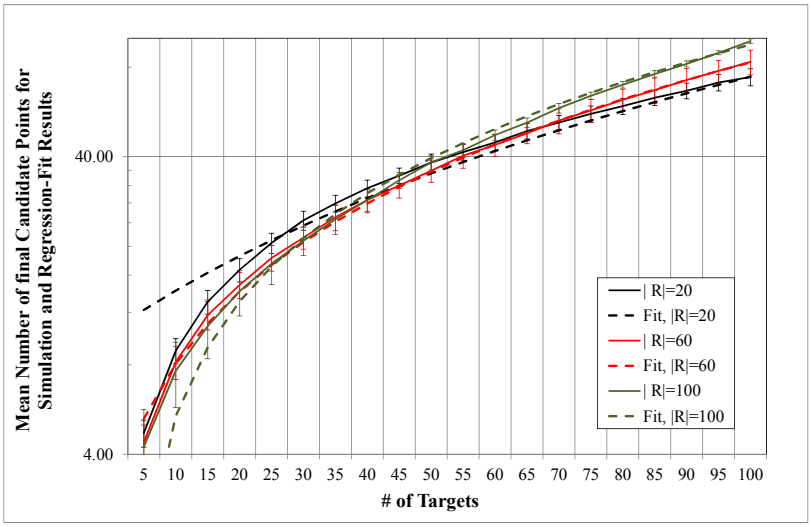


Figure 2.23: **Final Candidate Point Numbers from Simulation and Regression-Fit**— Number of candidate points for different numbers of receivers over the number of targets, where the simulations results and the one standard deviation error bars (solid lines) as well as the results using the fitted Equation 2.2 (dashed lines) are displayed.

$10 \times 10$ units and RoD of $\rho_0 = 0.6$, Equation 2.2 can be used to determine an approximation for the number of candidate points in the final list, given a specific number of targets and receivers.

Theoretical thoughts lead to the question of whether the increase of elements in the final target list flattens out or even decreases, if more and more targets are in a fixed area. This is implied because a limited AoI filled with more and more targets should lead at some point to a saturation, when additional CPs or IPs do not dominate existing areas. Moreover, a rising density of receivers and targets should lead to a small number of *super-dominant* candidate points which enable the detection of a very large number of targets and so leading to elimination of a huge amount of dominated candidate points. The reason is that the high density enforces the probability of a receiver-target combination lying very close to each other and so resulting in a very large detection disc encompassing a large number of smaller detection discs. Therefore, we conducted simulations with very high numbers of targets (up to 2,000) in order to verify or refute this conjecture. Because the calculation for this large number of targets exceeds the possibilities of a normal desktop computer, we use

the NPS HAMMING Cluster for these calculations. And even with this supercomputer the processing time of a single configuration set is high that only 10 to 25 replications for each design point are possible. A small set of the results is shown in Table 2.3. It shows the rapid

Table 2.3: Computing Time and Remaining Candidate Locations for High Target Numbers and Different Numbers of Receivers

| # targets | # receivers | # CPs Start | # CPs End | Computing Time [s] |
|---|---|---|---|---|
| 250 | 50 | 3,224 | 296 | 14.41 |
| 500 | 50 | 13,282 | 956 | 228.01 |
| 1,000 | 50 | 52,863 | 3,456 | 6,967.65 |
| 1,500 | 50 | 116,074 | 7,264 | 45,217.54 |
| 500 | 25 | 7,385 | 635 | 112.16 |
| 1,000 | 25 | 29,513 | 2074 | 2,169.11 |
| 2,000 | 25 | 120,741 | 7,651 | 56,454.89 |
| 100 | 100 | 954 | 95 | 1.21 |
| 250 | 100 | 5,701 | 425 | 34.90 |
| 500 | 100 | 22,592 | 1,483 | 636.48 |
| 1,000 | 100 | 91,172 | 5,583 | 21,367.80 |

growth of computing time with high numbers of targets. A doubling of $m$, while keeping all other parameters fixed, results in 20 to 40 times greater computing times. With this, even the supercomputer is not able to solve for example the configuration of $m = 1,800$ and $k = 50$ with ten replications in four days. Thus, the calculation of those parameter settings had to be aborted. For the outcomes we are able to solve, we do not see a decrease in the trial number of CPs

To investigate the effect with other means, in a last experiment, we vary the RoD instead of the number of targets. In a setting with $k = 25$ receivers and targets in the range $m = 100, 150, \ldots 300$, we consider $\rho_0 = 0.5, 1, \ldots 5$. The results can be seen in Figure 2.24. The plot shows clearly an increase of number of final and initial candidate points up to a specific level of the RoD and then a decrease. For the initial points this decrease starts at $\rho_0 = 2.5$ and is not very steep. This means that at a specific level of the RoD, the DDs become so large that multiple intersections between circles go down and the number of potential positions for sources decrease. For the final candidate points the decrease starts at $\rho_0 = 1.5$. The number drops quickly and reaches a level below 10 candidate points for $\rho_0 \geq 4$. This means that, starting with a RoD of two, larger DDs start to dominate smaller ones, leading to some candidate points which dominate a large amount of other candidate

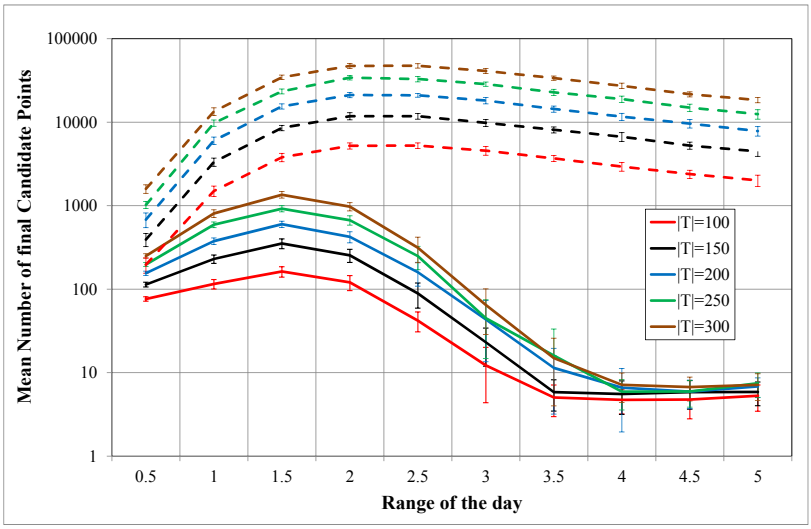


Figure 2.24: **Final Number of Candidate Points (solid lines) and Starting Number of Candidate Points (dashed lines) for Different Numbers of RoD**—Number of candidate points over RoD and standard deviation errorbars showing a decrease with larger RoDs.

points. For extreme values of a RoD greater than four, there are a few DDs that are so large that they include a major amount of the present candidate points for this set. As a result, only a few of these super-dominant points will be present after eliminating the dominated points. So the assumption that larger DDs lead to dominating points of higher order, for example, which detect a vast amount of targets, is proven, and so also a higher density of targets and recaivers—which has as a consequence larger DDs—would lead to this result, if the problem could be solved in an acceptable time.

Taking also the time for preprocessing into account (see Figure 2.25), one can see the interesting effect, that the computing time develops with rising RoD in a way between the two numbers of points with a slight tendency to the number of final points. For future work, a closer look could be taken into this relation to determine an estimation of the connection between preprocessing time, RoD, and the number of starting and final candidate points. This would be helpful to assess for a given configuration of sensors and targets the number of candidate points and expected computing before having to solve the whole problem. Thus it would be possible to identify problems which are not solvable in an affordable time with existing computer technology.

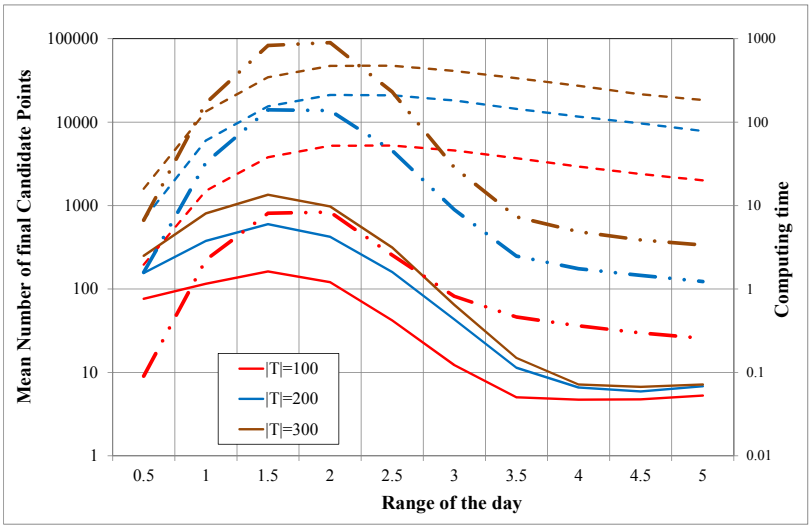


Figure 2.25: **Final Number of Candidate Points (solid lines), Starting Number of Candidate Points (dashed lines), and Preprocessing Time (dash-dotted) for Different Numbers of RoD**—Number of candidate points, resp. computing time over RoD.

# CHAPTER 3:
# Improved Placement of Sources and Receivers

This chapter deals with the improved placement of sources and receivers for a given set of targets—we present two main algorithms, Adapt-LOC and Iter-LOC, as well as their advantages as well as their limitations.

To make the results comparable, the general setting for all computational experiments in this chapter is always the same: in a square area with dimensions $10 \times 10 = 100$ units$^2$, a number of $m$ targets $t \in T$ are randomly placed. Given a RoD $\rho = 0.6$ and the target weights $0 \leq w_t \leq 1$ characterizing their value, the task is to place $k$ receivers $r \in R$ and $l$ sources $s \in S$ in a way that maximizes the expected value of the targets detected. In this setting, we compare the results obtained from our algorithm with those obtained by OPT-LOC and Greedy-LOC.

## 3.1 Source and Receiver Positioning Utilizing Detection Discs

The investigated approach is based on the usage of DDs and a further development of the LOC-GEN-II algorithm—the main procedure can be found in Figure 3.1 and is named Adapt-LOC. As a starting point, only the positions of the targets are known. Thus it is not possible to determine either the target-receiver distances or the radii of the resulting DDs. Therefore, in the first step, all target positions are encircled with DDs based on the RoD. Because a multistatic sonar system is used, a receiver within $\alpha \cdot \rho$, where $\alpha$ is a constant, can be used to detect the target. In a heuristic approach, we vary the radius of the DDs by a fixed factor $\alpha$, with the base case $\alpha = 1$ and $r_t = \rho$, as we calculate intersection points of all DDs. The non-dominated candidate points (either IPs or CPs) become possible receiver positions, and we use OPT-LOC or Greedy-LOC to determine the $k$ best of them. Now, we have a similar setting as in Chapter 2 and can use the same approach again (combination of LOC-GEN-II and OPT-LOC or Greedy-LOC) to find the optimal places for the $l$ sources. In this section we use Greedy-LOC to determine the locations of all sensors. The variation of $\alpha$ has, as a consequence, reduced target-receiver

distances and larger target-source distances ($d_{t,r} < \rho$ and $d_{t,s} > \rho$) for $\alpha < 1$ and vice versa $d_{t,r} > \rho$ and $d_{t,s} < \rho$ for $\alpha > 1$ . This algorithm can be used as a two-step process (see

0. Input parameters are: targets $T$, $x_t, \forall t \in T$, RoD $\rho$, $\alpha$
1. Set $d_t = \alpha \cdot \rho, \forall t \in T$, determine $d_{t,t'}, \forall t, t' \in T$
2. Create DDs $\delta_t \in D$, with center $x_t$ and radius $d_t, \forall t \in T$
3. For all pairs of DDs $\delta_t, \delta_{t'} \in D$ calculate all $I_{t,t'}$ and build $\bar{I} = \bigcup_{t,t' \in T} I_{t,t'}$
4. Execute LOC-GEN-II as written in Figure 2.6
5. Choose $k$ receiver locations using OPT-LOC or Greedy-LOC
6. Restart with LOC-GEN-II with the given sets of targets and receivers
7. Choose the $l$ source locations using OPT-LOC or Greedy-LOC

Figure 3.1: **Adapt-LOC to Find Best Places for Sources and Receivers Given a Set of Targets**—the main parts are highlighted in red

Figure 3.1) to place all receivers and sources. This is the base case computed in this thesis. Alternatively, the sources and receivers could be split into groups, with each group placed alternately until all are in place. The advantage of the second approach might be that the combination of sources and receivers, placed up to a specific step, could add benefit to the detection of additional targets in the next step, when placing the following subset of sources and receivers. But in this thesis we focus on the base case of the new algorithm; the further development can be done in future work.

The implementation of the base case algorithm in MATLAB can be found as a supplemental download at the NPS Dudley Knox Library. For the implementation, the order of placing the sensors is arbitrary. Accordingly we will use the terminology *sensor type 1* and *sensor type 2* from now on.

Although it is not important whether sources or receivers are placed first, the number of each sensor type is important. A placement of five sensors in the first step and ten in the second step will lead to a different configuration of the near optimal positions compared to the case of placing ten sensors first and five in the second step. The LOC-GEN-II algorithm in step six of Figure 3.1 bases the placement of the sensors of type 2 on the location of the targets and the $k$ sensors of type 1. $k = 5$ leads a different set of shortest distances and thus different radii of the DDs compared to $k = 10$, hence a different setting of candidate points, from which Adapt-LOC determines the best ones.

To get an overview of the algorithm and its performance for different configurations of

parameters, we use a nearly orthogonal Latin hypercube (NOLH) design, generated by a spreadsheet of Sanchez (2011), to keep the design points and parameter configurations for the first setting in a manageable size. Since all parameters $m, k, l$, and $\alpha$ can be varied, a NOLH design yields a good approximation of the output values across the whole range of the variables. We use the following ranges : $m \in [50, 150]$, $(k, l) \in [5, 50]$, and $\alpha \in [0.5, 1.5]$. The results (weighted percentage of detected targets and number of detected targets) can be seen in Table 3.1. The first configuration represents the mid-point of the setting. The following eight cases embody a situation when a smaller amount of receivers is placed at first and in a second step a larger amount of sources ($l > k$). These are subdivided in two groups with a starting target-receiver distance smaller then RoD (or $\alpha < 1$) on the one hand and $\alpha > 1$ on the other. For the second block of eight settings, this subdivision into two groups is equivalent, but now we look at a larger number of receivers and smaller number of sources ($k > l$).

Table 3.1: Percentage of detected targets based on weights and overall number with Adapt-LOC for different configurations of targets, receivers, sources, and mean distance of receivers to the targets compared to the random placement of the receivers with only optimal source placing (benchmark).

| **# T** | **# R** | **# S** | **Distance** | **Adapt-LOC** | **Benchmark** | **Adapt-LOC** | **Benchmark** |
|---|---|---|---|---|---|---|---|
| m | k | l | $\alpha$ | weighted | weighted | # targets | # targets |
| 100 | 28 | 28 | 1 | 0.938 | 0.813 | 0.847 | 0.694 |
| 150 | 13 | 33 | 0.81 | 0.786 | 0.667 | 0.6568 | 0.551 |
| 119 | 22 | 42 | 0.5 | 0.976 | 0.892 | 0.903 | 0.773 |
| 63 | 39 | 47 | 0.94 | 1 | 0.999 | 1 | 0.999 |
| 75 | 11 | 25 | 0.56 | 0.866 | 0.731 | 0.710 | 0.567 |
| 88 | 19 | 50 | 1.31 | 0.999 | 0.988 | 0.998 | 0.934 |
| 69 | 5 | 36 | 1.13 | 0.925 | 0.861 | 0.769 | 0.673 |
| 144 | 30 | 44 | 1.25 | 0.974 | 0.909 | 0.914 | 0.812 |
| 106 | 8 | 16 | 1.38 | 0.560 | 0.444 | 0.453 | 0.352 |
| 113 | 36 | 5 | 0.69 | 0.455 | 0.300 | 0.362 | 0.251 |
| 131 | 50 | 19 | 0.88 | 0.841 | 0.713 | 0.751 | 0.628 |
| 56 | 25 | 11 | 0.75 | 0.889 | 0.581 | 0.764 | 0.473 |
| 94 | 47 | 39 | 0.63 | 1 | 0.987 | 1 | 0.942 |
| 125 | 44 | 30 | 1.44 | 0.876 | 0.863 | 0.788 | 0.771 |
| 50 | 42 | 22 | 1.19 | 0.973 | 0.937 | 0.912 | 0.832 |
| 138 | 16 | 8 | 1.06 | 0.402 | 0.305 | 0.328 | 0.254 |
| 81 | 3 | 13 | 1.5 | 0.628 | 0.604 | 0.536 | 0.514 |

The standard deviations of the results are in the magnitude of 5% of the absolute values for the improved placement, with 500 runs of every configuration. A test run with only 100 replications leads to the same magnitude of the standard deviations. This can be explained by random placement of targets, so each experiment has a specific spatial target distribution which may or may not be well detectable. For example, a configuration where many targets are concentrated in a locally restricted area of the AoI will be easier to observe and so will result in a higher percentage of detected targets in comparison to the case of all targets being equally distributed over the whole area. So there is inherent variation in detection probability even with the improved algorithm. Additionally, the benchmark algorithm with randomly placed receivers has a higher standard deviation of the results (on average greater than 7%) as a result of further variation due to randomness in the receiver positions.

Nevertheless, as an overall result, it can be observed that improved placement of receivers and sources leads to better detection of the targets compared to the benchmark process and the results of both procedures can be separated from each other. But the differences between these two algorithms vary extremely for different parameter configurations with a minimum of 2% in absolute detection percentage and a maximum of even more than 30%. Settings where both algorithms allow detection of all targets (Table 3.1, rows 4, 6, and 12) are not further considered, because they do not allow differentiation between the algorithms' performance. As main trends, one can observe better performance with Adapt-LOC when the sensor-target distance for the first step is less or equal to the RoD, and when more sensors are placed in the first step (see the light-green-colored rows). The best performance is reached with $\alpha = 0.75$ and must sensors are placed in the first step; this leads to an increased detection percentage of 30.9%. Values of $\alpha > 1.4$ result in poorest improvement with a negligible increase of 2%, which is lower than the standard deviation for this experiment. But for $\alpha = 1.36$ Adapt-LOC performs well again; that is why it is not possible to set up clear rules regarding which parameter combination leads to significantly better results based on the NOLH designed configuration points. Therefore we conduct further experiments over the whole range of the parameters to get an answer to this question.

Figure 3.2 shows the percentage of weighted target detection of a series of experiments for $m = 150$ targets. Two random placement techniques are included in the graph: first the random placement of the type 1 sensors, used by Craparo and Karataş (2014) (denoted as

"Benchmark"), as well as complete random placement of both types of sensors, as used by Washburn and Karataş (2015). The entirely random placement was the first used algorithm and can so be seen as a base case. The graph reveals that all improved placements, even the benchmark with only placing sensor type 2 optimally, significantly outperform the base case. Detection rate of the benchmark algorithm is higher then the base case, but almost always below Adapt-LOC. Only $\alpha \geq 1.5$ results in fewer detected targets for some instances. The reason for this is that placing many type 1 sensors with a large distance to the targets

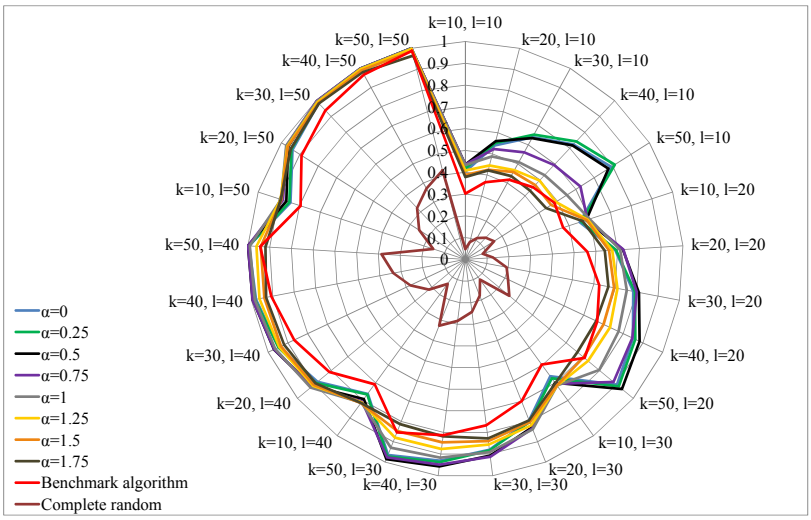


Figure 3.2: **Improved Positioning of Receivers and Sources Using Adapt-LOC**—Percentage of detected weighted targets for improved and random placement of receivers and optimal source placement. Both were placed at once, first the receivers and then the sources. $\alpha$ represents the width of the detection discs of the targets for the first step of Figure 3.1, the set has $m = 150$ targets, and the number of receivers ($k$) and sources ($l$) determines the investigated configuration.

($\alpha \gg 1$) in the first step has as a direct consequence the positioning of type 2 sensors in close vicinity to the targets to achieve detection. If the number of type 2 sensors is much lower than the number of type 1, only a few targets can be detected, because only a few targets have a type 2 sensor in sufficiently close distance.

Another relationship can be discovered between radii of target DDs and detection probability. A closer distance between target and first placed type 1 sensor always results in higher detection performance. The best outcomes can be found for $\alpha \in \{0, 0.25, 0.5\}$, which is also a trend in the NOLH designed experiments. The justification for this is that if many type 1 sensors are very close to targets, we have large DDs for the placement of type 2 sen-

Table 3.2: **Relation between Number of Different Sensors, Radius of Target Detection Discs and Detection Rate**

| **# sensors type 1** **k** | **# sensors type 2** **l** | **radius of target DDs** $\alpha$ | **detection rate** $\sum m_{detected} / \sum m$ |
|---|---|---|---|
| high | low | low | very high |
| low | high | low | low |
| high | low | high | very low |
| low | high | high | high |

sors. Since these DDs have larger radii, the possibility of achieving multiple intersections between them is higher, resulting in more candidate points for the optimal placement of type 2 sensors. Hence, $\alpha < 1$, up to a point, leads to better detection rates with an increasing number of type 1 sensors ($k$) while the type 2 sensor number ($l$) is fixed. Such being the case, when using Adapt-LOC to place sensors of both kinds (see Figure 3.1), one always has to consider how many sensors are to be placed in each step, as well as the range of the target-discs to find the best places for type 1 sensors. Table 3.2 summarizes the results. Overall performance of the settings starting with $\alpha \leq 0.75$ is much better than settings with larger values of $\alpha$; therefore, in the following, only $\alpha \leq 0.75$ will be taken into account. Figure 3.3 displays the results for this range of $\alpha$ with $m = 200$ targets. Again, Adapt-LOC shows significantly higher detection rates compared to the benchmark algorithm. With a standard deviation of around 5%, a clear distinction between the two kinds of placement is possible; because of that, the improved placement overwhelms the partially random placement. Up to 30% more targets can thus be detected by improved placement of both sensors, where $\alpha = 0.5$ shows the best overall performance. Two other visible trends in the graphs are the increased detection with a larger number of either type 1 sensors or type 2 sensors. More sensors lead to higher detection rate, as expected. We find that to achieve the highest increase of performance using Adapt-LOC, one should choose $\alpha \approx 0.5$. Ten more sensors of either increase the detection rate by approximately 10% for our instances. As consequence, additional sensors to a given setting should always be equally distributed between sources and receivers to gain the highest increase in performance.

Quantitative differences between Adapt-LOC and the benchmark can be seen in Figure 3.4. Given is a set of targets (×). In the left figure we see the candidate positions (•) and the best locations (○) for sensor type 1 (receivers) as chosen by Adapt-LOC. The middle plot

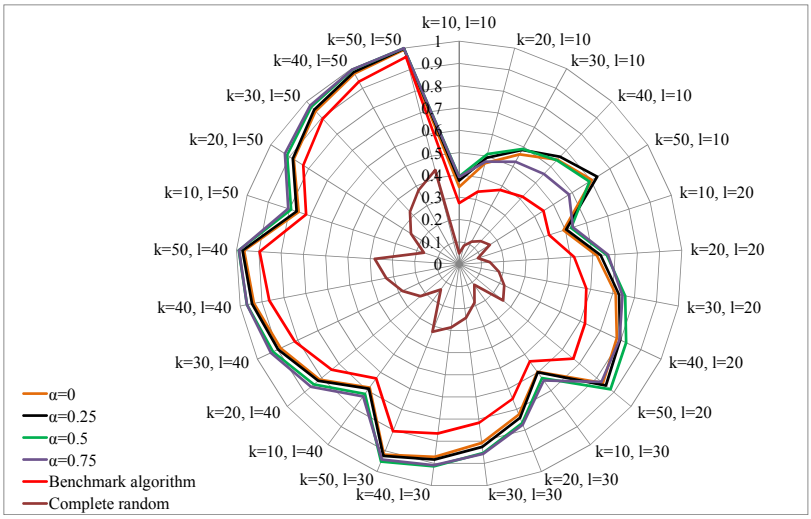


Figure 3.3: **Improved Positioning of Receivers and Sources using Adapt-LOC**—Percentage of detected weighted targets for improved and random placement of receivers and optimal source placement for $m = 200$ targets. Only the random placements and the improved placements for $\alpha \in \{0, 0.25, 0.5, 0.75\}$ are displayed.

shows the selected positions of the type 1 sensors (○), the candidate points for the positions of sensor type 2 determined by LOC-GEN-II (●), and the optimally placed sensors of type 2 (△). The right plot shows the targets (×) and randomly placed sensors of type 1 (○) and, for this configuration, the results for the optimal places of sensor type 2 (● and △). Adapt-LOC detects 80% of the targets in this example, compared to 50% for the benchmark. Looking at the structure of the result, Adapt-LOC shows a clustering of the type 1 sensors in the space between groups of targets. For example, the group around $(x, y) \approx (2, 3)$ shows four receivers in a group of nine detected targets, and only two sources are needed to achieve detection. As another example the detection of six targets at $(x, y) \in \{(4.5, 6.5), (0.5, 2.5)\}$ is possible with only one source and two receivers.

But Adapt-LOC also has disadvantages. As mentioned before, for specific values of $\alpha \geq 1.5$, the results become sometimes worse than the random placement of sensor type 1. Another drawback is shown in Figure 3.5 for the case if $\alpha = 1$. Radii of the DDs for type 1 sensor placement become $\rho_0$. With this intersection points of the DDs and therefore candidate points for sensor positioning are all $\rho_0$ apart from the target positions. When in the following step of Adapt-LOC the optimal positions of type 2 sensors are determined, the optimal distance of these sensors to a target will often be $\rho_0$, too. Thereby, sensor positions

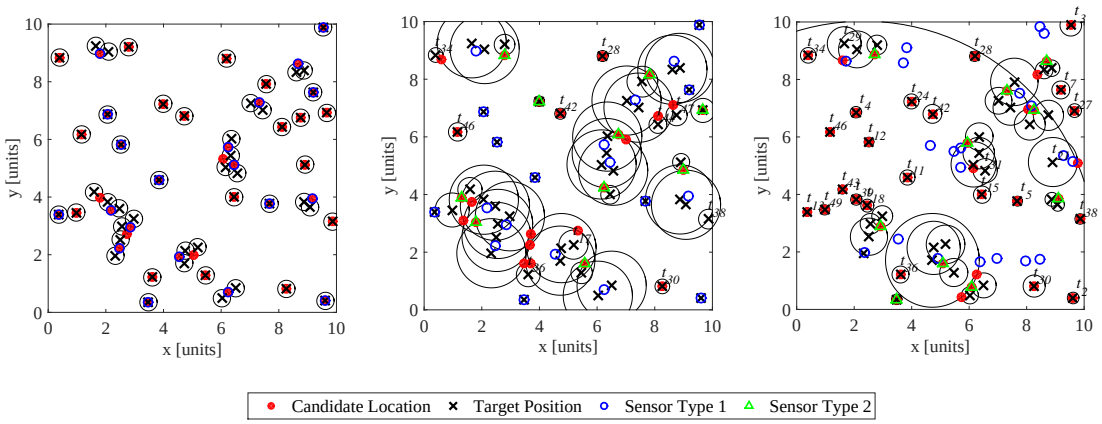


Figure 3.4: **Sensor Placement for Partially Random and Improved Position Determination**—Determination of optimal positions of type 1 sensors (left) for $m = 50$ targets, $k = 20$ receivers, $l = 10$ sources, and $\alpha = 0.5$. Greedy-LOC was used to determine the optimal positions of type 2 sensors (here source) for improved placement of type 1 sensors (middle) and randomly placed receivers (right). Adapt-LOC (left and middle graph) allows detection of 80% of all targets, the partially random one as benchmark of 50%. Non-detected targets are labeled with their number $t_x$.

of type 1 and type 2 coincide frequently. What is omitted this way is the advantage of multistatic detection systems to disperse receiver and source. In Figure 3.5, sensor positions of type 1 (○) and type 2 (△) coincide in nine of ten locations for Adapt-LOC (middle plot), and the radii of the DDs in the left and middle figure match very often. These disadvantages and the critical influence of individual parameter selection on the performance have to be taken into account using Adapt-LOC. But when the parameters are set appropriately, large improvements can be reached in comparison to the benchmark.

## 3.2 Near Locally Optimal Placement of Receivers and Sources

We now extend Adapt-LOC to produce a locally optimal placement using an iterative approaching—this further development is named Iter-LOC. As before, initially, only the position of the targets is known. In the first step, we determine for type 1 sensors for a specific value of $\alpha \in [0.25, 0.75]$ as in Adapt-LOC. In the second step, the optimal posi-

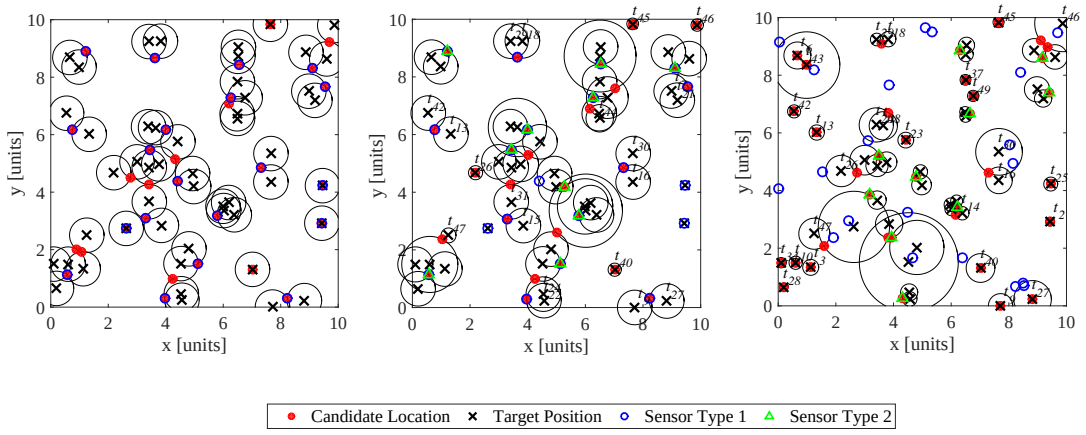


Figure 3.5: **Sensor Placement for Random and Improved Position Determination**— Determination of optimal positions of sensors of type 1 (left) for $m = 50$ targets, $k = 20$ receivers, $l = 10$ sources, and $\alpha = 1.0$. Greedy-LOC was used to determine the optimal positions of sensor of type 2 (here source) for improved placement of type 1 sensors (middle) and randomly placed receivers (right). Adapt-LOC (left and middle graph) allows the detection of 60% of all targets, the benchmark of 46%. Non-detected targets are labeled with their number $t_x$.

tions of type 2 sensors are determined on the basis of the target and type 1 sensor positions, again as in Adapt-LOC. After getting this first result, which is equivalent to the one in the last section, we now iterate to find locally optimal solutions. The positions of the type 1 sensors are erased, and OPT-LOC or Greedy-LOC is used to recalculate their optimal locations based on the placement of targets and type 2 sensors. Afterward, all type 2 sensors positions are erased and recalculated using OPT-LOC or Greedy-LOC, now for the given places of targets and type 1 sensors. This process is repeated until no further improvement can be reached with respect to target detection, where each iteration-step consists of two single sub-steps to calculate positions of both kinds of sensors. Thus, Iter-LOC begins with Adapt-LOC and proceeds using OPT-LOC or Greedy-LOC until no further improvement is possessive. As result one gets a locally optimal positioning of sources and receivers based on the location of targets. For the calculations in this section, we use Greedy-LOC to determine the optimal locations of sensors. The reason for this is so that only the MATLAB computing software is necessary and no additional software (GAMS) has to be used. This becomes important when the HAMMING cluster-computer is used and a call of GAMS out

of a MATLAB routine is not possible. Additionally, Iter-LOC uses a greedy-like approach to determine candidate positions and so provides the preprocessed data in a greedy-like format, such that the Greedy-LOC algorithm to determine near optimal locations should lead to faster solutions of the problem.

In Figure 3.6 we see the results of using Iter-LOC with $m = 100$ targets in addition to the results for all other algorithms discussed so far. A clear distinction between the outcomes of the four ways of setting the sensors can be seen. The lowest results represent the random placements (red and brown lines); these are always outperformed by Adapt-LOC when $\alpha \in [0.25, 0.75]$ (dashed lines). In these experiments the lower values of $\alpha$ perform better than $\alpha = 0.75$.

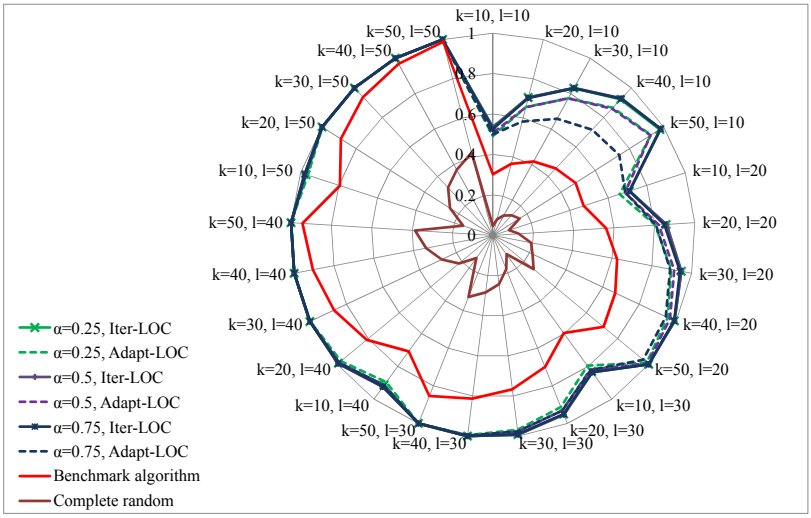


Figure 3.6: **Iterative Positioning of Receivers and Sources**—Percentage of detected weighted targets for Iter-LOC, Adapt-LOC, and random placement of receivers and sources for $m = 100$ targets. The iterative improved placements using Iter-LOC for $\alpha \in \{0.25, 0.5, 0.75\}$ are nearly congruent with each other.

Using Iter-LOC different values of $\alpha \in [0.25, 0.75]$, the detection rates nearly superimpose with each other (solid lines). This means that the starting value of $\alpha$ has only a negligible impact on the overall detection of targets when using Iter-LOC. This is reasonable because $\alpha$ mainly effects the first positioning of the sensors of type 1. But after the first full iteration of the local optimization process, these starting positions of the type 1 sensors get erased and a complete new set of locations for them are determined. Thus the influence of $\alpha$ on the detection rate is considerably reduced after the first iteration step. The results of Iter-

LOC outperform all the other methods in all configurations of parameters and also lead to an additional increase in target detection in comparison to Adapt-LOC, as can be seen in Figure 3.6. For example, with the configuration $k = 50$, $l = 10$, $\alpha \in [0.25, 0.5, 0.75]$, almost 98% of the weighted targets can be detected; the best result of Adapt-LOC was 92% for $\alpha \in [0.25,\ 0.5]$ and 0.74% for $\alpha = 0.75$. Thus, at least an additional 6% more targets can be detected using the same number of sensors. For the configuration $\alpha = 0.75$, this values jumps up to 24% more detected targets.

It can also be seen that the overall shape of the curves stays the same for Adapt-LOC and Iter-LOC. Peaks of detection can be found for the largest number type 1 sensors, when keeping the number of type 2 sensors constant and vice versa for the largest number of type 2 sensors for a constant type 1 sensor number. But the levels of detection for the iterative placement are significantly higher.

The detection values of Iter-LOC for different combinations of parameters ($m$, $k$, $l$, $\alpha$) can be found in Table 3.3.

To show the great advantage of Iter-LOC the whole random placement of all sensors and the benchmark with random placement of type 1 sensors as well as Adapt-LOC placement are also displayed. The benchmark algorithm is still the random type 1 sensor and optimal type 2 sensor placement.

The biggest improvements of Iter-LOC are highlighted in green and represent in excess of 12% more detected targets compared to Adapt-LOC. Compared to the benchmark algorithm, the increase in detected targets reaches almost 45%, which is an enormous enhancement. Using the iterative approach, we also eliminate the comparably inferior results of Adapt-LOC when choosing the "wrong" value for $\alpha$. The difference of the detection rate is less than 1% for all investigated values of $\alpha$. Using the iterative approach, one always gets locally optimal results regardless of which value of $\alpha$ had been chosen. The results for $m = 200$ targets can be seen in Figure 3.7 and show even more clearly the great advantage of Iter-LOC compared to random setting of type 1 sensor (solid orange line) and the Adapt-LOC method (dashed red line). The configuration $k = 50$ and $l = 10$ reflects the superior result of a more than 12% increase in target detection, which was also highlighted in Table 3.3.

Table 3.3: **Random, Partially Random, Adapt-LOC, and Iter-LOC Positioning of Receivers and Sources**—Percentage of detected weighted targets for different placement methods different number of targets.

| **# targets** | **# 1** | **#2** | **Random** | **Random** | **Improved** | **Iterative improved** | | |
|---|---|---|---|---|---|---|---|---|
| $m$ | $k$ | $l$ | pure | # 1 | $\alpha = 0.5$ | $\alpha = 0.25$ | $\alpha = 0.5$ | $\alpha = 0.75$ |
| 100 | 10 | 10 | 0.0452 | 0.3544 | 0.50660 | 0.535 | 0.5334 | 0.5256 |
| 100 | 50 | 10 | 0.1533 | 0.5435 | 0.9229 | 0.9777 | 0.9827 | 0.9824 |
| 100 | 50 | 30 | 0.3125 | 0.921 | 1 | 1 | 1 | 1 |
| 100 | 50 | 50 | 0.4092 | 1 | 1 | 1 | 1 | 1 |
| 150 | 10 | 10 | 0.0470 | 0.3022 | 0.4408 | 0.4603 | 0.4609 | 0.4631 |
| 150 | 30 | 10 | 0.1118 | 0.4171 | 0.6447 | 0.7228 | 0.7280 | 0.7253 |
| 150 | 50 | 10 | 0.1563 | 0.486 | 0.7762 | 0.8905 | 0.898 | 0.8924 |
| 150 | 10 | 30 | 0.1146 | 0.596 | 0.7019 | 0.7205 | 0.7232 | 0.7274 |
| 150 | 30 | 30 | 0.2427 | 0.767 | 0.9087 | 0.9432 | 0.9438 | 0.9404 |
| 150 | 50 | 30 | 0.3271 | 0.8536 | 0.9853 | 0.9990 | 0.9992 | 0.9986 |
| 150 | 10 | 50 | 0.1551 | 0.7951 | 0.8748 | 0.8758 | 0.888 | 0.8926 |
| 150 | 30 | 50 | 0.3242 | 09381 | 0.9967 | 0.9992 | 0.9995 | 0.9996 |
| 150 | 50 | 50 | 0.422 | 0.9881 | 1 | 1 | 1 | 1 |
| 200 | 10 | 10 | 0.0486 | 0.2744 | 0.393 | 0.4138 | 0.4161 | 0.4162 |
| 200 | 50 | 10 | 0.1627 | 0.4471 | 0.6929 | 0.8194 | 0.8325 | 0.8258 |
| 200 | 50 | 30 | 0.326 | 0.8055 | 0.9513 | 0.987 | 0.9877 | 0.9849 |
| 200 | 50 | 50 | 0.4352 | 0.9592 | 0.9997 | 1 | 1 | 1 |

To get an overview of the additional computer time needed to solve Iter-LOC, we show the number of steps needed to achieve a detection probability deviating less than 0.1% in the last two iterations in Figure 3.8. We determined this criterion empirically and find that it leads to a sufficiently good solution while still restricting the number of iterations. Lower limits did not lead to significantly better detection results but to a large increase in iterations. According to the procedure, the lowest number of steps must always be five, because each iteration consists of two steps and at least two iterations are needed to fulfill the criterion. That is why the curves never go below this value. The number of steps starts with a mean value around ten and lingers there for the different number of targets up to the configuration $k = 20$, $l = 30$. Then the values start to swing and reach a maximum of 20 to 30 for $k \in [20,30]$ and $l \in [40,50]$, and minimum values of five for $k \in [40,50]$ and $l \in [30,50[$. The smallest number of iterations is achieved when the target detection—even with Adapt-LOC—reaches nearly 100%, and so no further iterations are necessary. The highest number of iterations is needed in the "intermediate states" where the highest

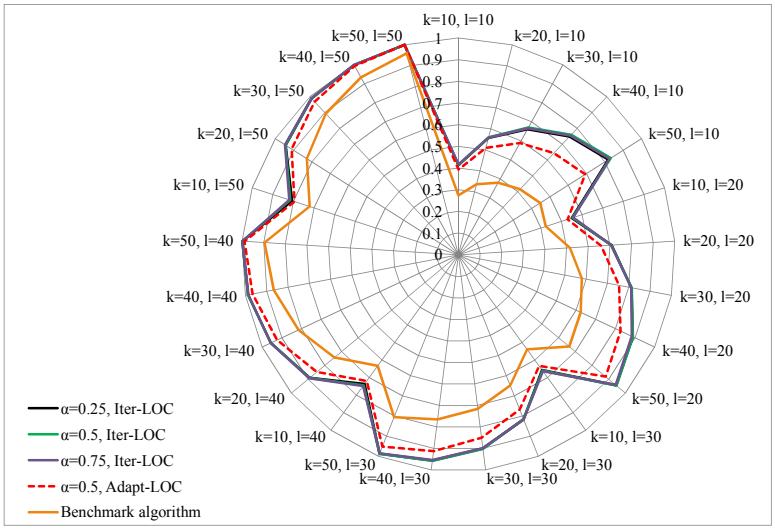


Figure 3.7: **Iterative Positioning of Receivers and Sources**—Percentage of detected weighted targets for Iter-LOC, Adapt-LOC, and random placement of receivers and sources for $m = 200$ targets. Iter-LOC placements for $\alpha \in \{0.25, 0.5, 0.75\}$ fall nearly apart.

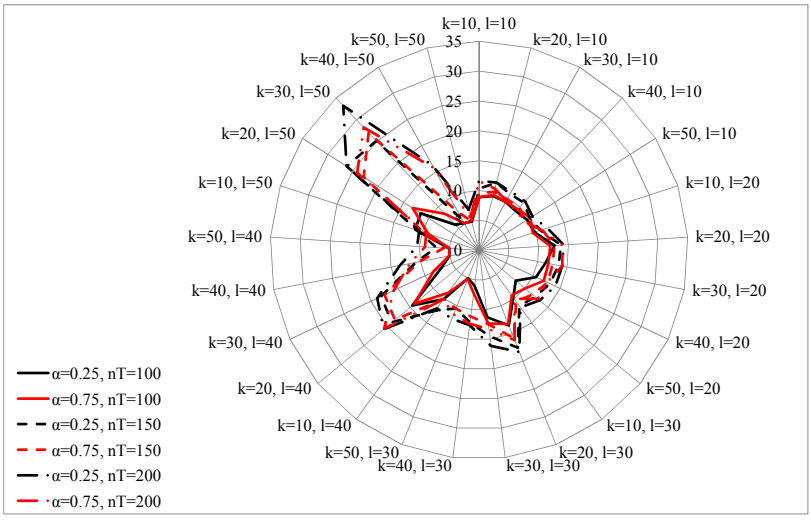


Figure 3.8: **Iterative Positioning of Receivers and Sources**—Mean number of steps until in the last four steps less than 0.1% change in the detected targets occurred; the minimum number of steps must thus be five.

increase in detection rate can be reached due to the iterative approach. This tendency can, for example, be observed for $l = 20$, where the iteration steps have a local maximum at $k = 50$, which also represents the configuration with the highest gain in target detection. This effect would also be expected for higher values of $l > 30$, but apparently the high

detection rate of Adapt-LOC overwhelms this effect leading to only a few iteration steps to reach near optimal detection probability. Overall it can clearly be seen that the number of iterations for this set of parameters stays in a computationally manageable order.

To estimate computational effort of Iter-LOC in comparison to the older methods and Adapt-LOC, the mean computing time for 100 replications of an optimal placement using random type 1 sensor placement, Adapt-LOC, and Iter-LOC are computed on the cluster computer of NPS. The configuration was restricted to $m \in \{100, 150, 200\}$, $(k,l) \in \{10, 20, \ldots 50\}$, and $\alpha = 0.5$. This is sufficient because at this place we only want to estimate the computing time of the new algorithms and find out whether they are computationally manageable or not. Generating more efficient versions of these new algorithms could be a task for further studies.

The computing times are depicted in Figure 3.9 and cannot be directly compared to the computing times in Section 2.5, because different computers were used for calculations. But the results allow an estimation whether Iter-LOC or the Adapt-LOC can be solved

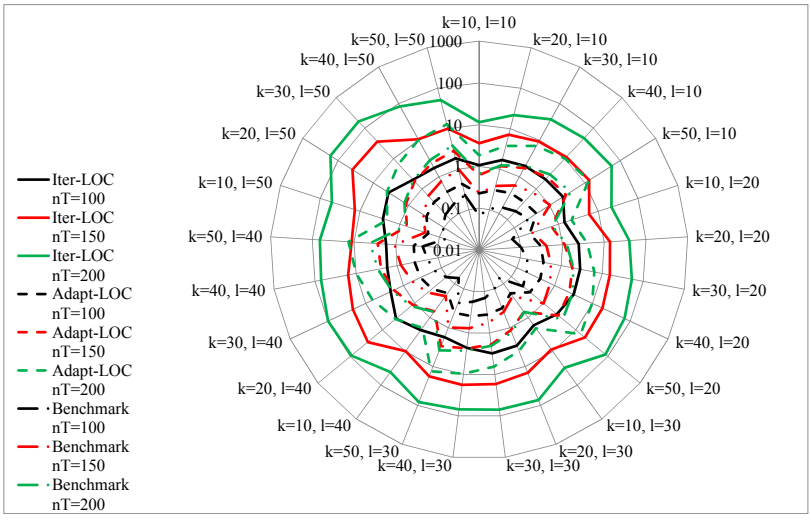


Figure 3.9: **Computing Time for Different Positioning Types of Receivers and Sources**— Mean computing time for $m \in \{100, 150, 200\}$, $(k,l) \in \{10, 20, \ldots 50\}$, and $\alpha = 0.5$ for random sensor type 1 (dashed dotted), Adapt-LOC (dashed), and Iter-LOC (solid).

in an acceptable time frame. As a benchmark we use random type 1 sensor placement, LOC-GEN-II for preprocessing and Greedy-LOC for optimal type 2 sensor positions; this appears as dashed-dotted lines. The Adapt-LOC placement of both sensor types (dashed

lines) needs approximately twice up to three times this reference time to solve the problems. But this is still acceptable, especially taking into account the significantly increased target detection rate with this new algorithm.

Switching to Iter-LOC, the computing time becomes, even for this set of parameters, exceedingly high. Compared to the benchmark algorithm, the computing is, in the case of higher numbers of iterations, up to 100 times larger than the random type 1 sensor placement. Compared with Adapt-LOC, on average a ten-folding of computing time takes place. Thus, here a clear tradeoff has to take place between improved target detection by up to 12% compared to Adapt-LOC, and the increased computing time. Especially in the case of high target numbers ($m \geq 200$) the computing time can become large—demonstrated by the more than 100 seconds computing time to solve one configuration on a cluster computer.

An interesting effect is observed when running Iter-LOC with Greedy-LOC as a subroutine: the optimal objective value does not always increase monotonically. This implies that a solution in one iteration step could lead to slightly worse results compared to the former one. Greedy-LOC delivers suboptimal solutions of the sensor placement problem. With this, a situation can occur where, for example, all recent source positions (based on the receiver positions of two steps back) are erased and newly calculated using detection discs based on the actual receiver positions. But these detection discs can lead to another set of best locations for the sources which might be a bit disadvantageous compared to the former set. To investigate the tradeoffs involved in using a greedy approach to sensor placement, we now use OPT-LOC as subroutine in Iter-LOC.

## 3.3 Exact Locally Optimal Placement of Receivers and Sources

One disadvantage of using Opt-LOC as a subroutine in Iter-LOC is that different software platforms have to be utilized. Due to restrictions of the cluster computer (it is impossible to run a MATLAB routine with a GAMS subroutine on the cluster), this experiment is conducted on a desktop computer with performance data as described in Chapter 2. This shows one shortcoming of using two different software platforms, because an outsourced solution of the problem on the faster cluster computer may not be possible. Computing times for identical problems solved on the cluster computer and on the desktop computer

grow by a factor of three in our experience, so computing times of the results in this section are not directly comparable to those of the last.

Figure 3.10 and Figure 3.11 compare the performance of Iter-LOC when Greedy-LOC and OPT-LOC are used as subroutines. One trend that is observable in the results is a declining performance in target detection with the following order of configurations: Iter-LOC with OPT-LOC, Iter-LOC with Greedy-LOC, Adapt-LOC with OPT-LOC, and Adapt-LOC with Greedy-LOC. As expected, the exact Iter-LOC algorithm outperforms all other algorithms without exception and results in the best set of detected targets. But Iter-LOC with Greedy-LOC is nearly as good. OPT-LOC allows up to 1% more detected targets compared with the near locally optimal solutions using Greedy-LOC for $m = 100$ targets (see Figure 3.10) and up to 2.5% for $m = 200$ targets (see Figure 3.11). The overall detection rate of the benchmark algorithm can thus be outperformed by more than 45.5%, which is a significant increase in performance. The greedy approach is, with up to 44%, not much inferior, but better results of the exact algorithm make that one the method of choice. It can also be seen

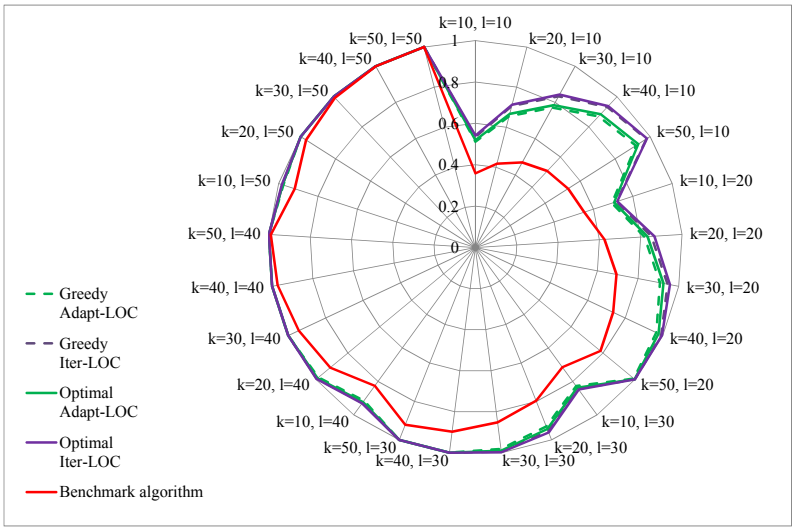


Figure 3.10: **Locally Optimal Positioning of Receivers and Sources Using Greedy-LOC and OPT-LOC**—Percentage of detected weighted targets for improved placements and random placement of receivers and sources for $m = 100$ targets. Exact locally optimal approach using OPT-LOC (solid lines) compared to near optimal locations using Greedy-LOC (dashed) and the benchmark (red line).

that for both optimizations the gap between Iter-LOC and Adapt-LOC becomes larger with increasing target numbers (see here especially Figure 3.11). Adapt-LOC achieves 14% less

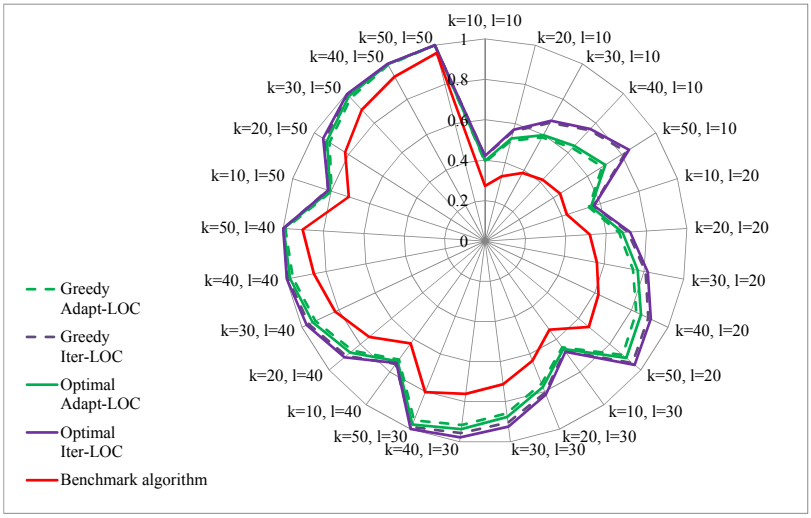


Figure 3.11: **Locally Optimal Positioning of Receivers and Sources Using Greedy-LOC and OPT-LOC**—Percentage of detected weighted targets for improved placements and random placement of receivers and sources for $m = 200$ targets. Exact locally optimal approach using OPT-LOC (solid lines) compared to near optimal locations using Greedy-LOC (dashed) and the benchmark (red line).

detection for a configuration with $k = 50$ and $l = 10$ compared to Iter-LOC. As a result Iter-LOC should always be preferred with respect to detection performance.

But also keeping computing time and iteration steps in mind, one sees great disadvantages of Iter-LOC, especially when using Greedy-LOC. As stated before, detection rates are not always increasing in successive iteration steps when using Greedy-LOC. Whereas, using OPT-LOC, each successive iteration step must always lead to better or at least the same performance. Solving the ILP cannot result in detection discs, which lead to decreased target detection in successive iterations and so detection rates are monotonically increasing. The outcome of this fact is a lower number of iterations, especially for larger target numbers, when using OPT-LOC (see Figure 3.12). The curves of the exact approach (solid lines) never exceed 14 steps and show a relatively smooth trend. On the contrary, when using Greedy-LOC we perform up to 37 steps and see distinct peaks for specific configurations of parameters (dashed lines).

This first drawback of using Iter-LOC with Greedy-LOC is underlined by the corresponding computing times, which are displayed in Figure 3.9 in the last section. Even on a cluster-

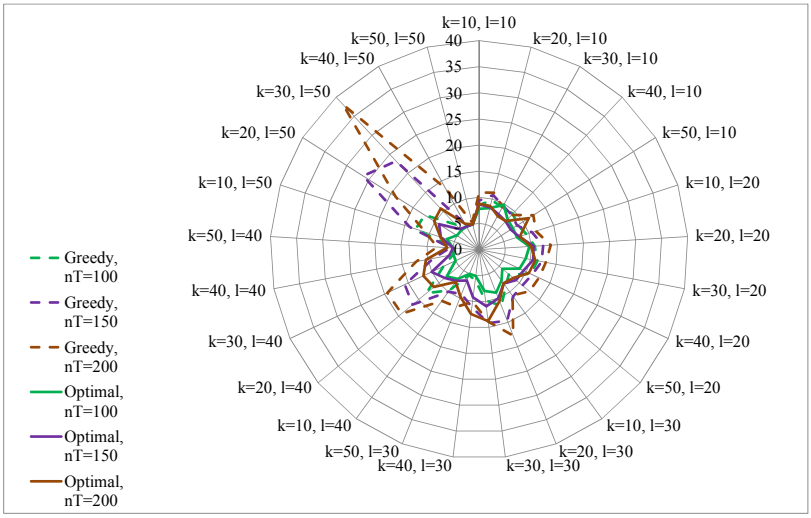


Figure 3.12: **Locally Optimal Iterative Positioning of Receivers and Sources**—Mean number of iterations until in the four following steps less than 0.1% change in the detected targets occurred; the minimum number of iteration steps must so be five.

computer a solution of moderate problems could need more than 160 seconds, which is a computing time more than 100 times larger than the benchmark. As stated previously, computing times are not directly comparable for different methods because different computer systems had been used, but Figure 3.13 shows better overall performance when using Iter-LOC with OPT-LOC. The multiplication factor to the benchmark performance goes down to 50 as maximum value and a problem solution can be achieved in less than 250 seconds on a standard computer. On a cluster-computer, which on average performs three times faster than the standard computer, this would be equivalent to 80 seconds for Iter-LOC with OPT-LOC and so half the computing time of Iter-LOC with Greedy-LOC.

This result is remarkable, because using Iter-LOC with OPT-LOC needs recurrent writing and reading of intermediate results in files by both software tools (GAMS and MATLAB). This is necessary in each iteration step and represents a very time consuming procedure. But even with this time consuming procedure Iter-LOC with OPT-LOC needs only half the equivalent computing time of Iter-LOC with Greedy-LOC. A further development of the connection and data handover between the two programs would have a great gain in performance as an implication and could be investigated in future studies. But even with reduced performance of Iter-LOC with OPT-LOC used in this thesis, it still outperforms

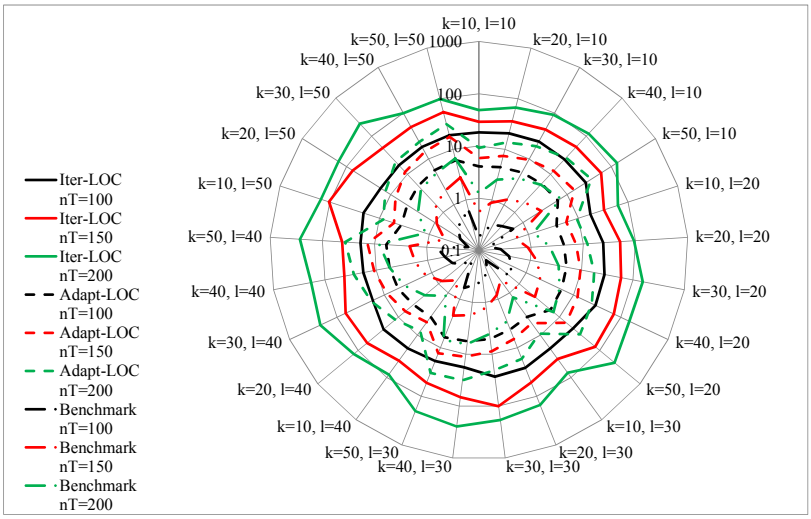


Figure 3.13: **Computing Time for Iterative Exact Locally Optimal Positioning of Receivers and Sources**—Mean computing time for $m \in [100, 150, 200]$, $(k,l) \in [10, 20, \ldots 50]$ and $\alpha = 0.5$ for random sensor type 1 (dashed dotted), Adapt-LOC (dashed) and Iter-LOC placement (solid).

in both cases, the numbers of iterations as well as the computing time itself, Iter-LOC with Greedy-LOC significantly. With this, the Iter-LOC with Greedy-LOC has only one advantage in comparison to the Iter-LOC with Opt-LOC: the restriction to one software platform. In all other categories (target detection and computing time in iterative and one-step approach, as well as number of iterations), it is always outperformed by Iter-LOC with OPT-LOC; therefore, the greedy approach should only be used if one has to stick to one software platform. In all other cases the Iter-LOC with OPT-LOC should be used as method of choice.

But even though the exact location determination leads to great detection values, a clear balancing of benefit (additional target detection) and expenditure (increased computing time) has to take place whether using Iter-LOC or Adapt-LOC. When the application is not time critical and the number of sensors and targets does not become too large ($m < 500$ targets, resp. $m + k + l < 500$), Iter-LOC should be used. But if the number of targets and sensors gets far beyond these values or the problem is time critical, the computational time of problem dissolution reaches unacceptable levels and the usage of cluster computers to solve the problems is inevitable. If cluster computing is not usable due to restrictions in operating GAMS out of MATLAB then this is not a viable solution. Therefore, future

work should also focus on implementing the solution of OPT-LOC in MATLAB instead of GAMS. This should be possible, because MATLAB offers also optimization toolboxes, but this work was beyond the scope of this thesis. The implementation of the OPT-LOC in MATLAB would make it possible to make all calculations on one software platform, which is applicable to the cluster computer and does not need successive writing and reading of files. As a result, overall performance of the exact locally optimal algorithm should thus be increased drastically.

# CHAPTER 4:
# Conclusion

Optimal sensor placement in multistatic environments is a mathematically highly complex problem. Researchers in the past have attempted this, usually facing the problem of unacceptably long computing times due to the complexity of the problem when multiple sensors have to be placed (Craparo & Karataş, 2014; Kuhn, 2014). Introducing DDs opens new possibilities to determine optimal sensor placement in multistatic sonar environments. The starting point for the preprocessing always has to be the determination of these points as either intersection points or center points of DDs. Sensor placement can only be optimal at non-dominated points, for which reason the key of successfully developing fast preprocessing algorithms is the theory of dominated and non-dominated points and the rapid separation of these two classes of points. Recent preprocessing uses this theory in its benchmark algorithm, but compares every candidate point with one another. Due to the high complexity of this algorithm, computing time becomes unacceptably large with greater problems.

We have developed the LOC-GEN-II algorithm, a fast method to solve this problem based on an iterative greedy-like approach. This process reduces the necessary calculations in each iteration step by determining the most dominant point and eliminating all of those candidate locations that are dominated by it. This leads to a reduction of computing time of up to 56%.

Following preprocessing, a sensor placement has to be found. We have investigated the possibility to reduce overall computing time of preprocessing and optimal sensor position by terminating the preprocessing at earlier stages and not investigating all possible candidate positions. LOC-GEN-II allows this proceeding based on the greedy-like approach being used for candidate point selection. Reduction of computing times is possible, but only in subordinate magnitude. Due to computationally higher complexity of this combined processing and the comparably low performance gain, it is not recommended to be used.

Additionally, we have investigated the number of final candidate points in the reduced candidate point list. We observed a relationship between target number and number of final candidate locations. The class of the point (IP or CP of DD) has the most significant influence on the trend of the preprocessing time for different numbers of targets. Increasing numbers of targets lead to an ascending number of final candidate points. Using regression, we are able to make very good predictions for several combinations of numbers of targets and sensors. Analyzing the influence of the detection range (RoD) leads to the interesting result of a decreasing number of candidate points for larger values of RoD.

Continuing on, we have developed an algorithm to deploy both kinds of sensors for a given set of targets. The benchmark was the recent algorithm which allowed optimal positioning of sources for a given set of targets and receivers, both of which are already in place. Thus, placement of both types of sensors in multistatic systems has not been possible in a reasonable time up to now. Based on LOC-GEN-II, we use the concept of DD to solve the optimization problem in a two-step or iterative approach. For a set of targets we use an adapted LOC-GEN-II algorithm to identify locations for the receivers. Based on this choice, we determine optimal source locations using LOC-GEN-II as an existing placing algorithm. Compared to the benchmark algorithm, we obtain considerably better detection results, and it is possible to increase the number of detected targets by up to 30%.

We have also developed an iterative approach for placing both types of sensors. In the first step, we execute the calculation stated in the paragraph above. Starting from this first guess, in each iteration step the positions of type 1 sensors are erased and recalculated based on the positions of targets and type 2 sensors from the previous step. With these new positions of type 1 sensors and targets, we calculate improved locations for the placement of type 2 sensors. This ends the first iteration step, and we repeat the process until only slight enhancements of detection are possible. This process leads to improved detection probability of up to 44% compared to the benchmark and, additionally, clearly outperforms the one-step algorithm. For both placement processes (one-step and iterative), near optimal greedy solutions as well as locally optimal ILP solutions are compared. Interestingly, for the iterative approach, ILP outperforms the greedy approach in terms of iteration steps, though not necessarind in terms of computing time.

For a higher number of targets, computing time grows rapidly. This leads to problems,

especially for the iterative approach, and shows the limitations of the developed algorithms. Another limitation is the fact that the iterative algorithms only guarantees a locally optimal solutions, not globally optimal.

For further analysis globally optimal solutions could be developed, with the restriction of solving the problem in an adequate time. Another possibility is implementing the ILP solution into MATLAB and so restricting the whole calculation process on one software platform. Up to now, the preprocessing has taken place in MATLAB and the optimization in GAMS, which requires time consuming calls of the different programs and storing temporary results in files in each step. Combining both algorithms in one software platform (e.g., MATLAB) should therefore lead to a significant reduction of computing time. Another interesting work with respect to the preprocessing part could be a closer look into the relation between possible candidate point number, final candidate point number, and RoD and their relation with respect to the configuration of the experimental setting.

THIS PAGE INTENTIONALLY LEFT BLANK

# APPENDIX A:
# Calculation of Intersection Points of Detection Discs

Assume two DDs $\delta_i$, center $(x_i, y_i)$, radius $r_i$ and $\delta_j$, center $(x_j, y_j)$, radius $r_j$. The distance between their centers can be calculated via $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ and these discs intersect if $d_{ij} \leq r_i + r_j$ and they overlap if $d_{ij} < r_i + r_j$. The intersection points $(x_{ij}, y_{ij})$, resp. $(x'_{ij}, y'_{ij})$ of the DDs can than be calculated using the following formulae (Craparo & Karataş, 2014, p. 16):

$$x_{ij}, x'_{ij} = \frac{x_i + x_j}{2} + \frac{(x_j - x_i)(r_i^2 - r_j^2)}{2d_{ij}^2} \pm \frac{(y_j - y_i)}{2d_{ij}^2} \sqrt{\left((r_i + r_j)^2 - d_{ij}^2\right)\left(d_{ij}^2 - (r_i - r_j)^2\right)} \tag{A.1}$$

$$y_{ij}, y'_{ij} = \frac{y_i + y_j}{2} + \frac{(y_j - y_i)(r_i^2 - r_j^2)}{2d_{ij}^2} \pm \frac{(x_j - x_i)}{2d_{ij}^2} \sqrt{\left((r_i + r_j)^2 - d_{ij}^2\right)\left(d_{ij}^2 - (r_i - r_j)^2\right)} \tag{A.2}$$

THIS PAGE INTENTIONALLY LEFT BLANK

# APPENDIX B:
# MATLAB Implementation Of LOC-GEN-II

```
clc
clear all
%-------------------------------------     PARAMETERS TO BE CHANGED
    ---------------------------------------%
m=10;               % length of area A
n=10;               % width of area A
nS=10; % 5:5:50     % Numbers of Sources - also declare in line 20
nT=50; % 20:20:200 % Starting number of targets - also declare in line 21
nR=50;              % number of receivers = default 50
numtrials=100;      % number of trials to run = default 100
rho=0.6;            % range of the day = default 0.6
plotgraph=0;        % plotgraph = 1 to activate plot
%-------------------------------------     PARAMETERS TO BE CHANGED (END)
    ---------------------------------%

DATA=zeros(numtrials*length(nS)*length(nT),14);
AverageResults=zeros(length(nS)*length(nT),13);
ind=0;
data_ind=0;

for nS=10 % 5:5:50                % Number of sources
    for nT = 50; % 20:20:200      % Number of targets
        ind = ind + 1;
        % Generate all needed Matrizes to make the calculation faster
        covered_greedy=zeros(numtrials,1);
        greedy_time=zeros(numtrials,1);
        opt_time=zeros(numtrials,1);
        covered_opt=zeros(numtrials,1);
        percent_covered=zeros(numtrials,1);
        NumberofCPsBefore=zeros(numtrials,1);
        NumberofCPsAfter=zeros(numtrials,1);
        createCP_time=zeros(numtrials,1);
        sum_W=zeros(numtrials,1);
        coverage_ratio=zeros(numtrials,1);
        % Run through the number of trials for each congifuration to get statistical
            significant results
        for trial = 1 : numtrials
            nS
            nT
            trial
            %Set initial values to zero
            DistTR=zeros(nT,nR);                % target-receiver distance matrix
            DistTT=zeros(nT,nT);                % target-target distance matrix
```

```
41          TargetsVisible = zeros(nT^2,nT);     % matrix with targets visible from
                intersection points
42          TargVisBinComp = ones(nT^2,nT);      % binary matrix: ones = target not visible
                 from intersection point
43          NumberIntersection = zeros(nT^2,4); % matrix for internal relation of
                detection disc numbers
44          NumIntersections = zeros(nT,1);      % vector with # of intersection points of
                each detection disk
45          radius=zeros(nT,nR);                 % radius of contours for each target and
                receiver set
46          PoD=zeros(nT*2,nT);                  % PoD of each target by a receiver for
                every CP
47          CPReduced=zeros(nT,2);               % matrix of reduced CP-set, maximum # of
                not diminated CPs is nT
48
49          % random coordinates for receivers
50          R=rand(nR,2);R(:,1)=R(:,1)*m;R(:,2)=R(:,2)*n;
51          % random coordinates for targets
52          T=rand(nT,2);T(:,1)=T(:,1)*m;T(:,2)=T(:,2)*n;
53          % random weights for targets
54          W=rand(1,nT);sum_W(trial)=sum(W);
55
56          % create the target-receiver distance matrix and radius matrix
57          for t=1:nT
58              for r=1:nR
59                  DistTR(t,r)=sqrt((R(r,1)-T(t,1))^2+(R(r,2)-T(t,2))^2);
60                  radius(t,r)=rho^2./DistTR(t,r);
61              end
62          end
63          max_radius=max(radius,[],2);
64          % create the target-target distance matrix
65          for ti=1:nT
66              for tj=1:nT
67                  DistTT(ti,tj)=sqrt((T(ti,1)-T(tj,1))^2+(T(ti,2)-T(tj,2))^2);
68              end
69          end
70
71          tic
72          CountTotInter = 0;
73          % Set all target positions as possible canditate points
74          ReduceCenter = T;
75          % find the intersection points of all disks
76          for int1=1:nT-1 % target A
77              for int2=int1+1:nT % target B
78                  % compute the (x1,y1) and (x2,y2) intersection points of disks
79                  x1=(T(int2,1)+T(int1,1))/2 + (T(int2,1)-T(int1,1)) * (max_radius(int1)
                        ^2-max_radius(int2)^2)/(2*DistTT(int1,int2)^2) - (T(int2,2)-T(int1
                        ,2))/(2*DistTT(int1,int2)^2) * sqrt(((max_radius(int1)+max_radius(
                        int2))^2-DistTT(int1,int2)^2)*(DistTT(int1,int2)^2-(max_radius(
```

```
    int1)-max_radius(int2))^2));
80  y1=(T(int2,2)+T(int1,2))/2 + (T(int2,2)-T(int1,2)) * (max_radius(int1)
    ^2-max_radius(int2)^2)/(2*DistTT(int1,int2)^2) + (T(int2,1)-T(int1
    ,1))/(2*DistTT(int1,int2)^2) * sqrt(((max_radius(int1)+max_radius(
    int2))^2-DistTT(int1,int2)^2)*(DistTT(int1,int2)^2-(max_radius(
    int1)-max_radius(int2))^2));
81  x2=(T(int2,1)+T(int1,1))/2 + (T(int2,1)-T(int1,1)) * (max_radius(int1)
    ^2-max_radius(int2)^2)/(2*DistTT(int1,int2)^2) + (T(int2,2)-T(int1
    ,2))/(2*DistTT(int1,int2)^2) * sqrt(((max_radius(int1)+max_radius(
    int2))^2-DistTT(int1,int2)^2)*(DistTT(int1,int2)^2-(max_radius(
    int1)-max_radius(int2))^2));
82  y2=(T(int2,2)+T(int1,2))/2 + (T(int2,2)-T(int1,2)) * (max_radius(int1)
    ^2-max_radius(int2)^2)/(2*DistTT(int1,int2)^2) - (T(int2,1)-T(int1
    ,1))/(2*DistTT(int1,int2)^2) * sqrt(((max_radius(int1)+max_radius(
    int2))^2-DistTT(int1,int2)^2)*(DistTT(int1,int2)^2-(max_radius(
    int1)-max_radius(int2))^2));
83  % if the intersection points are real (means they actually intersect)
    appoint them to the matrix IP.
84  % If a disk has intersection points, the center point of the disk will
     never become a candidate point,
85  % because it will always be subdominant to one of the points on the
    edge
86  if (isreal(x1)==1) && (isreal(y1)==1) && (isnan(x1)==0) && (isnan(y1)
    ==0) && (isinf(x1)==0) && (isinf(y1)==0)
87      % Increase the counter for the intersections of both disks related
         to the target pair
88      NumIntersections(int1) = NumIntersections(int1) + 1;
89      NumIntersections(int2) = NumIntersections(int2) + 1;
90      % Because an intersection exists, erase the center point from the
        Candidate Point List of the targets
91      ReduceCenter(int1,:)= zeros;
92      ReduceCenter(int2,:)= zeros;
93      % Increase the counter of the overall number of intersections
94      CountTotInter = CountTotInter + 2;
95      % Fill matrix with targets numbers and coordinates of interscetion
         points for further calculation
96      NumberIntersection(CountTotInter-1,1:2) = [int1,int2];
97      NumberIntersection(CountTotInter,1:2) = [int1,int2];
98      NumberIntersection(CountTotInter-1,3:4) = [x1,y1];
99      NumberIntersection(CountTotInter,3:4) = [x2,y2];
100     % Fill matrix of visible targets from each intersection point with
         visible target number (int3) and set binary element to ZERO (
        visible)
101     for int3 = 1 : nT
102         if sqrt((x1-T(int3,1))^2+(y1-T(int3,2))^2)<max_radius(int3)
              +0.00001
103             TargetsVisible(CountTotInter-1,int3) = int3;
104             TargVisBinComp(CountTotInter-1,int3) = 0;
105           end
```

```
106                         if sqrt((x2-T(int3,1))^2+(y2-T(int3,2))^2)<max_radius(int3)
                                +0.00001
107                             TargetsVisible(CountTotInter,int3) = int3;
108                             TargVisBinComp(CountTotInter,int3) = 0;
109                         end
110                     end
111                 end
112             end
113         end
114         % Erase all rows in matrixes for discs with NO intersactions
115         TargetsVisible(all(TargetsVisible==0,2),:) = [];
116         TargVisBinComp(all(TargVisBinComp==1,2),:) = [];
117         NumberIntersection(all(NumberIntersection==0,2),:) = [];
118         % Eliminate all rows in the Candidate Point List of the targets with
              intersections
119         ReduceCenter(all(ReduceCenter==0,2),:) = [];
120         % Starting number of candidate points
121         NumberofCPsBefore(trial) = size(ReduceCenter,1) + CountTotInter
122
123         % Now generate the Reduced Candidate Point List
124         % FIRST STEP: Find the Candidate Points of the discs with NO intersection
125         % A target position is a Candidate Point, if there is NO other disc without
              intersection in the disc
126         % OR
127         % there is another disc without intersection in the disc AND the inner disc
              does not include the center point of the outer disc
128         IndexZeros = find(~NumIntersections);
129         CPreducedCounter = 0;
130         for i = 1:length(IndexZeros)
131             Flag = 1;
132             for j = 1:length(IndexZeros)
133                 if DistTT(IndexZeros(i),IndexZeros(j)) < max_radius(IndexZeros(i)) &&
                        DistTT(IndexZeros(i),IndexZeros(j)) > max_radius(IndexZeros(j))
134                     Flag = 0;
135                 end
136             end
137             if Flag == 1
138                 CPreducedCounter = CPreducedCounter + 1;
139                 CPReduced(CPreducedCounter,:) = ReduceCenter(i,:);
140                 % Build up the PoD Matrix needed for the greedy algorithm : If CP Y
                      sees target X, then element X in the column Y of the POD matrix is
                       set to ONE
141                 PoD(CPreducedCounter,IndexZeros(i)) = 1;
142             end
143         end
144         StartIntersectionPoints = CPreducedCounter + 1;
145
146         % SECOND STEP - iteratively:
```

```
147         % A - find the Intersection Point which detects the most targets (most
                dominant), put it in the Reduced Candidate Point List
148         % B - eliminate all intersection points which are dominated by the point from
                step A from the Candidate Point List
149         % Repeat A and B as long there are intersection points in the Candidate Point
                List (= elements in matrix 'TargetsVisible')
150         laeufer2 = 0;
151         while(~isempty(TargetsVisible))
152             laeufer2 = laeufer2 + 1;
153             Targvis(laeufer2) = size(TargetsVisible,1);
154             % Find the first most dominant intersection point
155             dominant = find(max(nT-sum(TargVisBinComp,2))==(nT-sum(TargVisBinComp,2))
                    ,1);
156             dominantBin = TargVisBinComp(dominant,:);
157             % determine which targets can be detected from this intersection point
158             VisElm = TargetsVisible(dominant,:);
159             VisElm(all(VisElm==0,1))=[];
160             % Add the dominant intersection point to the Reduced Candidate Point List
161             CPreducedCounter = CPreducedCounter+1;
162             CPReduced(CPreducedCounter,:) = NumberIntersection(dominant,3:4);
163             % Generate the PoD Matrix endtry for the intersection point
164             for loopcounter = 1 : length(VisElm)
165                 PoD(CPreducedCounter,VisElm(loopcounter)) = 1;
166             end
167             % Loop through all remaining intersection points.
168             % Find all dominated intersection points and erase the related entries
169             % A Point is DOMINATED if it sees the same or fewer targets then the
                    DOMINANT point
170             % BUT it must not see any other target, not visible for the dominant point
171             for subdominant = 1 : size(TargetsVisible,1)
172                 if (sum(VisElm == find(TargetsVisible(subdominant,:)~=0,1))==1)
173                     if (sum(dominantBin .* TargetsVisible(subdominant,:))==0)
174                         NumberIntersection(subdominant,:) = zeros;
175                         TargetsVisible(subdominant,:) = zeros;
176                         TargVisBinComp(subdominant,:) = ones;
177                         NumIntersections(find(TargetsVisible(1,:)~=0,1)) =
                                NumIntersections(find(TargetsVisible(1,:)~=0,1))-1;
178                     end
179                 end
180             end
181             % Clear out the related elms to this dominant point from the working data
                    to find the next most dominant intersection point
182             NumberIntersection(dominant,:) = zeros;
183             TargetsVisible(dominant,:) = zeros;
184             TargVisBinComp(dominant,:) = ones;
185             NumIntersections(VisElm(1)) = NumIntersections(VisElm(1))-1;
186             % Eliminate all empty rows in the matrices
187             TargetsVisible(all(TargetsVisible==0,2),:) = [];
188             TargVisBinComp(all(TargVisBinComp==1,2),:) = [];
```

```
189                 NumberIntersection(all(NumberIntersection==0,2),:) = [];
190             end
191             % Eliminate all empty rows in the matrices
192             CPReduced(all(CPReduced==0,2),:) = [];
193             PoD(all(PoD==0,2),:) = [];
194             NumberofCPsAfter(trial) = CPreducedCounter;
195             createCP_time(trial) = toc;
196
197     %---------------------------------- End of new algorithm to calculate the Canditate
        Point List ----------------------------------%
198
199     %-------------------------------------     CREATE THE TXT FILES FOR K, a and P
        ---------------------------------------%
200             fid_D = fopen('MATLAB_D.txt','w');
201             for i=1:size(PoD,1)   %number of candidate points
202                 for t=1:nT        %number of targets
203                     if PoD(i,t)==1
204                         P_str=strcat('CP',num2str(i),'.t',num2str(t));
205                         fprintf(fid_D, '%s\r\n', P_str);
206                     end
207                 end
208             end
209             fclose(fid_D);
210
211             fid_t = fopen('MATLAB_t.txt','w');
212             for t=1:nT
213                 t_str=strcat('t',num2str(t));
214                 fprintf(fid_t, '%s\r\n', t_str);
215             end
216             fclose(fid_t);
217
218             fid_r = fopen('MATLAB_r.txt','w');
219             for r=1:nR
220                 r_str=strcat('r',num2str(r));
221                 fprintf(fid_r, '%s\r\n', r_str);
222             end
223             fclose(fid_r);
224
225             fid_s = fopen('MATLAB_s.txt','w');
226             for s=1:nS
227                 s_str=strcat('s',num2str(s));
228                 fprintf(fid_r, '%s\r\n', s_str);
229             end
230             fclose(fid_s);
231
232             fid_CP = fopen('MATLAB_CP.txt','w');
233             for i=1:size(CPReduced,1)+1
234                 CP_str=strcat('CP',num2str(i));
235                 fprintf(fid_CP, '%s\r\n', CP_str);
```

```
236             end
237             fclose(fid_CP);
238
239             fid_W = fopen('MATLAB_W.txt','w');
240                 for t=1:nT
241                     W_str=strcat('t',num2str(t),32,num2str(W(t)));
242                     fprintf(fid_W, '%s\r\n', W_str);
243                 end
244             fclose(fid_W);
245
246             if plotgraph==1
247                 display('plotting')
248                 % PLOTS THE CIRCLES AROUND POINTS OF INTERESTS
249                 % T(i,1) and T(i,2) are the (x,y) coordinates of the center of the circle
                        i
250                 % r(i) is the radius of the circle i
251                 % 0.05 is the angle step
252                 figure(1)
253                 subplot(1,2,1)
254                 cstring='rgbcmyk'; % color string
255                 for t=1:nT
256                     ang=0:0.05:2.1*pi;
257                     xp=max_radius(t)*cos(ang);
258                     yp=max_radius(t)*sin(ang);
259                     plot(T(t,1)+xp,T(t,2)+yp,'k');
260                     hold all
261                 end
262                 subplot(1,2,1)
263                 scatter(CPReduced(:,1),CPReduced(:,2),65,'o','MarkerfaceColor','red','
                        MarkerEdgeColor','r');
264                 scatter(R(:,1),R(:,2),60,'^','MarkerEdgeColor','b','MarkerFaceColor','b','
                        LineWidth',1.5);
265                 scatter(T(:,1),T(:,2),70,'x','MarkerEdgeColor','k','LineWidth',2.0);
266                 labels3 = num2str((1:size(CPReduced,1))','c_{%d}');
267                 text(CPReduced(:,1), CPReduced(:,2), labels3, 'horizontal','left', '
                        vertical','top','FontAngle','italic','FontSize',14,'Fontname', 'Times
                        New Roman');
268                 labels2 = num2str((1:size(R,1))','r_{%d}');
269                 text(R(:,1), R(:,2), labels2, 'horizontal','center', 'vertical','top','
                        FontAngle','italic','FontSize',14,'Fontname','Times New Roman');
270                 labels1 = num2str((1:size(T,1))','t_{%d}');
271                 text(T(:,1), T(:,2), labels1,'horizontal','left', 'vertical','bottom','
                        FontAngle','italic','FontSize',14,'Fontname','Times New Roman');
272                 axis square
273                 axis tight
274                 axis ([0 m 0 n]);
275                 hold off
276                 % subplot 2 contains the number of remaining CPs in the candidate list in
                        each step
```

```
277                 % using the algorithm to eliminate subdominant candicate points
278                 subplot(1,2,2)
279                 plot(Targvis);axis square; xlabel('Iteration step'); ylabel('Number of
                        remaining candidate points');
280                 hold off
281             end
282             display(' ALL DONE... ')
283
284     % Greedy algorithm ------------------------------------------------------
285             PdHOLD=PoD;
286             weighted_PdHOLD=PdHOLD.*repmat(W,size(PdHOLD, 1),1);
287             tic
288             covered=zeros(1,size(PoD,1));
289             selected=zeros(1,nS);
290             for k=1:nS
291                 % find CP that covers the most targets
292                 covered(:)=sum(weighted_PdHOLD,2);
293                 % choose it
294                 bestCP=find(covered==max(covered));
295                 covered_greedy(trial)=covered_greedy(trial)+max(covered);
296                 choose=bestCP(1); % index of candidate point selected
297                 targetscovered=find(PdHOLD(choose,:)==1);
298                 % remove targets associated with the CP that we just chose
299                 weighted_PdHOLD(:,targetscovered)=0;
300             end
301             greedy_time(trial)=toc;
302     % End greedy algorithm --------------------------------------------------------
303     % call GAMS to solve optimally
304             %! GAMS cookie_MIP.gms
305             ! c:\GAMS\win64\24.2\GAMS cookie_MIP.gms
306             load z.dat;
307             load comptime.dat;
308
309             opt_time(trial)=comptime;
310             covered_opt(trial)=z;
311             percent_covered(trial)=covered_greedy(trial)/covered_opt(trial);
312             coverage_ratio(trial)=covered_opt(trial)/sum_W(trial);
313             data_ind=data_ind+1;
314             DATA(data_ind,:)=[data_ind nS nT nR NumberofCPsBefore(trial) NumberofCPsAfter(
                        trial) createCP_time(trial) opt_time(trial) greedy_time(trial) covered_opt
                        (trial) covered_greedy(trial) percent_covered(trial) sum_W(trial)
                        coverage_ratio(trial)];
315         end
316         average_NumberofCPsBefore=mean(NumberofCPsBefore);
317         average_NumberofCPsAfter=mean(NumberofCPsAfter);
318         average_createCP_time=mean(createCP_time);
319         average_greedy_time=mean(greedy_time);
320         average_opt_time=mean(opt_time);
321         average_covered_greedy=mean(covered_greedy);
```

```
322         average_covered_opt=mean(covered_opt);
323         average_percent_covered=mean(percent_covered);
324         worst_greedy_performance=min(percent_covered);
325         average_coverage_ratio=mean(coverage_ratio);
326 
327         AverageResults(ind,:)=[nS nT nR average_NumberofCPsBefore average_NumberofCPsAfter
                average_createCP_time  average_opt_time average_greedy_time
               average_covered_opt average_covered_greedy average_percent_covered
               worst_greedy_performance average_coverage_ratio];
328         AverageResults(ind,:)
329     end
330 end
331 
332 save Complete_Solution_Base DATA AverageResults
```

**LOC-GEN-II Implementation** - This code shows the LOC-GEN-II implementation in MATLAB. We used the implementation of Craparo and Karataş (2014) as frame and implemented the new preprocessing algorithm. The algorithm to find the optimal solutions using GAMS or the near optimal greedy approach was completely taken from the algorithm of Craparo and Karataş (2014).

THIS PAGE INTENTIONALLY LEFT BLANK

# Supplementals

- Computer code of the LOC-GEN-II algorithm implemented in MATLAB—to be used with Appendix A
- Computer code of the Adapt-LOC and Greedy-LOC algorithm and their subroutines implemented in MATLAB

The supplementals are available at Dudley Knox Library of the Naval Postgraduate School in Monterey, CA.

THIS PAGE INTENTIONALLY LEFT BLANK

# References


Bowen, J. I., & Mitnick, R. W. (1999). A multistatic performance prediction methodology. *Johns Hopkins APL Technical Digest*, *20*(3), 424–431.

Braw, E. (2014, October). The "Russian Submarine" in Swedish waters isn't the only unwelcome visitor in the Baltic Sea. *Newsweek*. Retrieved from http://www.newsweek.com/2014/10/31/damaged-submarine-spotted-swedish-waters-russia-turns-baltics-278694.html.

Coon, A. C. (1997). Spatial correlation of detections for impulsive echo ranging sonar. *Johns Hopkins APL Technical Digest*, *18*(1), 105–112.

Coraluppi, S., & Grimmet, D. (2003). Multistatic sonar tracking. In I. Kadar (Ed.), *Proceedings of SPIE—Signal processing, sensor fusion and target recognition* (pp. 399–408). Orlando, FL: SPIE.

Craparo, E. M., & Karataş, M. (2014). *Sensor optimization in multistatic underwater sensor networks*. Manuscript submitted for publication.

Daerden, L. (2015, January). Full list of incidents involving Russian military and NATO since March 2014. *The Independent UK*. Retrieved from http://www.independent.co.uk/news/world/europe/full-list-of-incidents-involving-russian-military-and-nato-since-march-2014-9851309.html.

DelBalzo, D. R., McNeal, D. N., & Kierstead, D. P. (2005). Optimized multistatic sonobuoy fields. In *Proceedings of Oceans 2005—Europe* (pp. 1193–1198). IEEE.

DelBalzo, D. R., & Stangl, K. C. (2009). Design and performance of irregular sonobuoy patterns in complicated environments. In *Proceedings of Oceans 2009, MTS/IEEE Biloxi* (pp. 1–4). IEEE.

Fewell, M. P., & Ozols, S. (2011, June). *Simple detection-performance analysis of multistatic sonar for anti-submarine warfare* (Tech. Rep. No. DSTO-TR-2562). Edinburgh, South Australia: Defence Science and Technology Organisation.

Gong, X., Zhang, J., Cochran, D., & Xing, K. (2013). Barrier coverage in bistatic radar sensor networks: Cassini oval sensing and optimal placement. In *Proceedings of the Fourteenth ACM International Symposium on Mobile ad hoc Networking and Computing* (pp. 49–58).

Johnsen, T., & Olsen, K. E. (2006). *Bi- and multistatic radar* (Tech. Rep. No. RTO-EN-SET-086). Neuilly-sur-Seine, France: NATO Research and Technology Organization (RTO).

Kuhn, T. U. (2014). *Optimal sensor placement for point coverage in active multistatic sonar networks*, M.S. thesis, Naval Postgraduate School, Monterey, CA.

Marcus, J. (2014, November). Russia's "close military encounters" with Europe documented. *British Broadcasting Corporation*. Retrieved from http://www.bbc.com/news/world-europe-29956277.

Ngatchou, P. N., Fox, W. L., & El-Sharkawi, M. A. (2006). Multiobjective multistatic sonar sensor placement. In *Proceedings of 2006 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2713–2719). IEEE.

Orlando, D., & Ehlers, F. (2011). *Advances in Multistatic Sonar*. INTECH Open Access Publisher.

Ozols, S., & Fewell, M. P. (2011, June). *On the design of multistatic sonobuoy fields for area search* (Tech. Rep. No. DSTO-TR-2563). Edinburgh, South Australia: Defence Science and Technology Organisation.

Rosen, K. (2011). *Discrete Mathematics and Its Applications*. New York, NY: McGraw-Hill.

Sanchez, S. M. (2011). *NOLH Designs Spreadsheet.* Retrieved from http://harvest.nps.edu/.

Simeone, N. (2014, November). Hagel outlines budget reducing troop strength, force structure. *United States Department of Defense*. Retrieved from http://www.defense.gov/news/newsarticle.aspx?id=121703.

United States Navy. (2014, March). Other procurement, Navy. In *Department of Defense fiscal year (FY) 2015 budget estimates* (Vol. 3). Washington, DC: United States Navy.

Urick, R. J. (1983). *Principles of Underwater Sound* (3rd ed.). New York, NY: McGraw-Hill.

Washburn, A., & Karataş, M. (2015). Multistatic search theory. *Military Operations Research (MOR) Journal*, *20*(1), 21–38.

# Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California